SEP 0 8 2004
PATENT & TRADEMARK OFFICE
JC68

61 AF

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

DO

Atty. Ref.: 2687-8

Serial No. 09/412,334

TC/A.U.: 2126

Filed: October 5, 1999

Examiner: Anya, Charles

For: ARRANGEMENT FOR SIMPLIFYING THE DESIGN AND IMPLEMENTATION OF MOBILE SERVICES IN A COMMUNICATION SYSTEM

RECEIVED
SEP 1 0 2004
Technology Center 2100

* * * * * * * * * * * *

September 8, 2004

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

SUBMISSION OF PRIORITY DOCUMENTS

It is respectfully requested that this application be given the benefit of the foreign filing date under the provisions of 35 U.S.C. §119 of the following, a certified copy of which is submitted herewith:

Application No.	Country of Origin	Filed
1997 1605	Norway	08 April 1997

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
John R. Lastova
Reg. No. 33,149

JRL:at
1100 North Glebe Road, 8th Floor
Arlington, VA 22201-4714
Telephone: (703) 816-4000
Facsimile: (703) 816-4100

KONGERIKET NORGE
The Kingdom of Norway

Bekreftelse på patentsøknad nr
Certification of patent application no

1997 1605

▷ Det bekreftes herved at vedheftede dokument er nøyaktig utskrift/kopi av ovennevnte søknad, som opprinnelig inngitt 1997.04.08

▷ *It is hereby certified that the annexed document is a true copy of the above-mentioned application, as originally filed on 1997.04.08*

2004.08.10

Line Reum

Line Reum
Saksbehandler

PATENTSTYRET®
Styret for det industrielle rettsvern

Søknad om patent.

Søknadsskriv

1a - b

Styret for det industrielle rettsvern

Dato	Patentsøknad nr.
08.APR97	971605

Til
Patentstyret
Boks 8160 Dep.
0033 Oslo

Utfylles av Styret

- Patentsøknad nr.
- Inngivelsesdag
- Alment tilgjengelig
- Behandlende medlem EH
- Int. Cl.[6] H04L

JGS/bf

Søkers/fullmektigs referanse (angis hvis ønsket):

121110

Saksbehandler: J.G. Standberg

Alm.tilgj - 9 OKT. 1998

8.apr97 550739 P971605 1 0,00

Oppfinnelsens benevnelse:

Anordning for forbedring av tilgjengelighet av tjenester i et kommunikasjonssystem

Hvis søknaden er en internasjonal søknad som videreføres etter patentlovens § 31:

Den internasjonale søknads nummer .....

Den internasjonale søknads inngivelsesdag .....

Søker:
Navn, bopel og adresse. (Hvis patent søkes av flere. Opplysning om hvem som skal være bemyndiget til å motta meddelelser fra Styret på vegne av søkerne).
(Fortsett om nødvendig på neste side)

Telefonaktiebolaget LM ERICSSON
S-126 25 Stockholm
Sverige

Oppfinner:
Navn og (privat-) adresse.
(Fortsett om nødvendig på neste side)

Do Van Thanh
Stjernemyrvn. 28
0673 Oslo

Fullmektig:

Oslo Patentkontor AS, Postboks 7007 M, N-0306 Oslo

Hvis søknad tidligere er inngitt i eller utenfor riket:
(Fortsett om nødvendig på neste side)

Prioritet kreves fra dato ingen sted ..... nr. .....

Prioritet kreves fra dato..... sted ..... nr. .....

Prioritet kreves fra dato..... sted ..... nr. .....

Prioritet kreves fra dato..... sted ..... nr. .....

Hvis avdelt søknad:

Den opprinnelige søknads nr.:.....og dennes inngivelsesdag .....

Hvis utskilt søknad:

Den opprinnelige søknads nr.:..... begjært inngivelsesdag.....

971605

Angivelse av tegnings-figur som ønskes publisert sammen med sammendraget

Fig. nr. .....

POSTADRESSE	KONTORADRESSE	TELEFON	TELEKS	TELEFAKS	POSTGIRO	BANKGIRO
Boks 8160 Dep. 0033 Oslo	Københavngt. 10 Oslo	22 38 73 00	19 152 nopat n	22 38 73 01	0808 5170709	1600.40.39916

Bl. 10 - 10 000. 1-97. Sem Grafisk

Table of Contents

List of Figures

Table of Contents

List of Figures

1

Introduction

1.1 BACKGROUND

The telecommunications area is currently facing major and rapid changes. On the one hand, progressing liberalisation and monopoly abolition allow new actors to enter the scene and hence increase competition. On the other hand, the demand from the users for new and more sophisticated services such as interactive multimedia services and broadband services is also accelerating and introducing new opportunities and enormous potentials in the telecommunications market. The telecommunications industry needs, therefore, to cut down both the cost and the time to design, implement, introduce, maintain and extend telecommunications services. The traditional approach to the specification and design of telecommunication applications based on proprietary solutions and with low level of modularity does not adequately meet the new needs.

Fortunately, the concurrent evolution in technology enables the use of concepts, principles and methodologies from the computing world. With higher processing capability, larger primary and secondary storage devices, better switching and transmission technologies offering more bandwidth and better quality of service, the difference between a wide area telecommunications network and a local computer network disappears. The same applies to the distinction between a telecommunications application and a computing application. The telecommunications and the computing worlds are converging. Technologies, methodologies and experiences are exchanged and shared between the two worlds. Researchers in distributed computing are experimenting with digital switching hardware and new communications capabilities while developers in telecommunications are using de facto standard operating systems and middleware infrastructure in the construction of advanced services.

Therefore, It is natural that the concepts and principles defined in Open Distributed Processing (ODP) [ITUa], [ITUb], [ITUc] are adopted and used in telecommunications systems, bearing in mind that these systems are probably the largest and most complex of all existing distributed systems. The Reference Model of Open Distributed Processing (RM-ODP) is, however, not specific enough for telecommunications applications. TINA-C (Telecommunications Information Networking Architecture Con-

sortium) [TIN95i], [TIN94c] combines the ODP framework with the concepts defined for Intelligent Networks (IN) [CCI92a] and Telecommunication Management Network (TMN) [CCI92b], and defines a consistent and open architecture for telecommunications software applications. In TINA, a telecommunication application is specified as a set of interacting objects relying on a Distributed Processing Environment (DPE). The DPE, through its support of distribution transparencies, hides the complex aspects caused by distribution. The application designers do not need to be aware of the mechanisms necessary to deal with the different aspects of distribution and can therefore focus on their application specifications. The design of telecommunication applications is hence alleviated considerably.

Recently, the rapid expansion of digital cellular telephone expresses an urgent demand for mobility from the users and the necessity of the mobility support for telecommunications system [Kat94]. Both ODP and TINA-C define the mobility support as one important requirement. However, both ODP and TINA-C assume in their original framework that the support of mobility is considered as a consequence of the support of the other distribution transparencies and, therefore, does not require any additional functions and mechanisms in the DPE. In other words, mobility should be seamlessly supported if other distribution transparencies such as access and location transparencies are supported.

In this thesis we show that access and location transparencies are not sufficient for supporting terminal and personal mobility. We propose instead to consider mobility as an additional transparency, and in order to do so, we propose a Functional Separation Architecture where the mobility functions are separated both from the Network layer and the Application (Service) layer [TIN95i] and grouped into a separate layer called the Mobility layer.

A Generic Mobility System (GMS) is proposed as a mean to realise the Mobility layer. The GMS will be designed and then introduced in the architecture transparently, i.e applications although supported by the GMS are not aware of its existence. In this way all applications designed for fixed networks will be supported by the mobile system without requiring redesign or adaptation. However, we will also show that there are two other families of applications which explicitly uses the functionality of the GMS. The first family consists of applications which are required the mobility management such as a user registration. The second family consists of applications which actively use the location information held by the GMS. The GMS may be implemented and offered as a middleware[1] which can be customised, configured and installed in any type of networks and distributed systems in order to support the mobility transparency. A simplified implementation of the proposed GMS is under development at UNIK, the Center for Technology at Kjeller.

1. Ovum reports defines middleware as: off-the-shelf connectivity software which supports distributed processing at run-time and which is used by developers to build distributed software [RE95] [ring95:distObj].

1.2 The TELEcom REsearch Program TELEREP

The researched work described in this thesis constitutes the mobility part of the TEL-Ecom REsearch Programme TELEREP, carried out at the Center for technology at Kjeller, UNIK [Spi96]. Here is a short description of the programme:

The programme has been established to obtain practical experience with ODP/TINA methods and the implementation of TMN and IN functionality, and to study how security could be provided in the DPE environment. The programme started in 1995 and will run through several years. Initially the programme decided to base its work on a public domain version of CORBA [Obja], and will be re-evaluated at a later stage. One is currently studying both theoretically and practically, as part of a Ph.D programme, how mobility and security can be provided transparently by the platform environment.

Mobility is the capability to deliver telecommunication services to the user anywhere and at any time. With a DPE supporting the distribution transparencies defined by ODP/TINA, the design of distributed applications is not more complex than that of centralised applications. The mechanisms supporting the defined transparencies, however, are not sufficient to support mobility. This part of the research programme study the mechanisms required to support mobility and how such functionality can be integrated in the ODP/TINA system. With such mechanisms in place, telecommunication services such as telephone, videophone and videoconferencing and computer applications like word processor, speadsheet and mail, mobility will be completely transparent to the applications. This means that existing applications will be available to mobile users without any modifications and new applications can be designed in the same manner as regular "fixed" applications. This part of the programme is well under way and will likely be rounded off in first quarter of 1997. This thesis is the main contribution to this part of the programme.

Security as a study item in the context of ODP is in an early phase, with the main thrust in 1997.

Security in monolithic computer systems and in data communications are relatively mature fields. However, when monolithic systems are connected by data communications to form distributed systems, the security does not scale properly to provide protection to the distributed system as a whole.

ODP embodies a platform-based approach to distributed systems which shifts the focus from the combined monolithic and communication view to a distribution-independent application-driven view. With the platform approach, the view of security should also be application driven. But little work has been done so far on ODP security.

This research work aims at providing a model for ODP security that offers security transparently to applications, and with transparent enforcement by the platform of application security specifications. The relationship between security specifications and the platform security will be described with formal verifiable methods.

1.3 CLAIMS

The main contributions of this thesis are:

1. to show that mobility can be designed as an ODP transparency
2. to demonstrate that this transparency can be realised in a Functional Separation Architecture as a separate middleware called Generic Mobility System.

In order to achieve these goals we:

- demonstrate that the access and location transparencies cannot alone support terminal mobility and personal mobility.
- propose a Functional Separation Architecture supporting mobility
- propose a Generic Mobility System realising the Mobility layer of this architecture.
- develop and design the required mechanisms for terminal mobility support.
- develop and design the required mechanisms for user mobility support.
- develop the required mechanisms for the mobility-related security functions.
- introduce an approach by which mobility is made transparent to applications.
- propose guidelines regarding how applications are integrated into the telecommunications system.
- develop the required mechanisms for session mobility support
- introduce service interface for mobility-based application
- show that the GMS can be implemented.

1.4 OUTLINE OF THE THESIS

The thesis is organised as follows:

- Chapter 2 gives a brief presentation of ongoing work in the area of basic Open Distributed System: the concepts and principles defined in the RM-ODP, the TINA architecture and a short introduction to CORBA, the Common Object Request Broker.
- Chapter 3 presents some related work where the activities at TINA-C concerning mobility will be summarized. This part also gives an overview of the PCS TINA auxiliary project and the Dolmen project.
- Chapter 4 introduces the Generic Mobility System. Mobility is specified using the Enterprise and Information viewpoints of ODP. It is demonstrated that mobility is not supported in an ODP/TINA environment. A Functional Separation Architecture supporting mobility is presented and a Generic Mobility System supporting mobility transparently is proposed.

- Chapter 5 contains the detailed design of the terminal mobility support. The necessary functions are identified and the required computational objects are introduced gradually in the GMS.
- Chapter 6 describes the user mobility support. Additional computational objects are found and added to the GMS.
- Chapter 7 considers some mobility related security issues.
- Chapter 8 treats the problem of transparency. Three alternative ways to integrate the GMS into the system are considered and compared. One of the alternatives are proposed to be the preferred one.
- Chapter 9 study how the applications are supported by the GMS and how they can be integrated in the system. The notion of session is explained. The applications are divided into Incoming and Outgoing categories and it is considered how each of them are supported by the platform. The support of session mobility are also described. The chapter is rounded off by a description of the services offered by the GMS to the mobility-based applications.
- Chapter 10 gives an example of implementation of the GMS at UNIK, the Center for Technology at Kjeller.
- Chapter 11 concludes our work by reviewing and discussing the fufilment of claims of this thesis. Open problems are identified and areas for future work are suggested.

2

Open Distributed Systems

2.1 INTRODUCTION

In this thesis, the Open Distributed System paradigm is used as foundation for all discussions and developments. Open distributed system is a category of distributed systems which are a made up of components that may be obtained from a number of different sources, which together work as a single distributed system [Cro96]. In 1988, the International Standards Organisation (ISO) began work on preparing standards for Open Distributed Processing (ODP). These standards have now largely been completed, and they define the interfaces and protocols to be used in the various components of an open distributed system. The ODP standards assume a model where distributed applications are running in multiple processes in multiple computers linked by communications. The application designer will be supported by a programming environment and run-time system that will make many aspects of distribution in the system transparent. For instance, the designer may not have to worry about where the parts of the applications are running; this is called location transparency.

There is another approach to supporting applications in a distributed system, namely by using a distributed operating system. A program may be run on any computer in the distributed system and can access data on any other computer through an enhanced system interface. The advantage of a distributed operating system is that the existing programming environments may be used and also, in some cases existing software. The disadvantage is that a number of problems are left for the programmer and user to handle, for instance concurrency. Another disadvantage is that the distributed system is tied to a style of operating system interface. There are lots of different operating system today meeting different requirements and it is not possible or practicable to unify them by building a distributed operating system on the top of them. The distributed operating system solution is not used in this thesis.

The ODP model provides an application interface to the distributed system. This interface is simple and is concerned with aspects of distribution only. The application may run on any local operating system which is appropriate. In distributed computing, this model is called the virtual mainframe and is enabled by distributed object

technology [RC95]. The virtual mainframe is created from "componentware" which are collaborative suites of self contained, re-usable components, pre-fabricated and linked dynamically. The image of a monolithic mainframe is hence restored in the heterogeneous distributed system. This virtual mainframe is hence as simple to program and as robust as a real mainframe.

This chapter starts with a description of ODP since the ODP concepts are central in this thesis. The TINA architecture (Telecommunications Information Networking Architecture) which is a specialisation of ODP for telecommunications applications will be presented next. Some specifications of TINA such as the network resource management and the session concept are adopted but the other specifications of the service architecture are found unsuitable and hence omitted. Finally, the Common Request Broker Architecture (CORBA), an industry standard for open distributed processing being developed by the Object Management Group (OMG), will be covered. CORBA is used as platform in the test implementation at the Center for technology at Kjeller.

2.2 OPEN DISTRIBUTED PROCESSING (ODP)

The Open Distributed Processing framework is defined by the Reference Model of Open Distributed Processing (RM-ODP) [ITUa], [ITUb], [ITUc], [ITUd]. The model describes an architecture within which support of distribution, interworking, interoperability and portability can be integrated.

The most basic concepts defined are those of object, action and interaction; an object encapsulates its state, and its states can only be modified by interaction with other objects or by the internal actions of the object itself. The interactions between objects take place at interfaces; an object may have multiple interfaces.

There are two main structuring approaches used in the ODP architecture: the definition of viewpoints and the definition of transparencies [Lin94].

2.2.1 Viewpoints

Distributed system can be very large and complex and the RM-ODP defines five viewpoints for the specification. The five viewpoints are:

- the **enterprise viewpoint** focuses on the purpose, scope and policies for the system.
- the **information viewpoint** focuses on the semantics of the information and information processing performed.
- the **computational viewpoint** enables distribution through functional decomposition of the system into objects which interact at interfaces.
- the **engineering viewpoint** focuses on the mechanisms and functions required to support distributed interaction between objects in the system.
- the **technology viewpoint** focuses on the choice of technology in that system.

The viewpoints are not completely independent; key items in each are identified as related items in other viewpoints. The viewpoints can be related to the software engineering process as shows in Figure 2.1.

The development process used in our research work is a little different. We also start with an enterprise viewpoint specifying the actors involved in mobility. But in the second phase, the functional specification and the design are combined and the information and computational viewpoints are modelled in parallel, revisited several times and expanded gradually until the design is completed. The engineering viewpoint is very simple since the mechanisms necessary for distribution are supposed to be supported by the infrastructure.

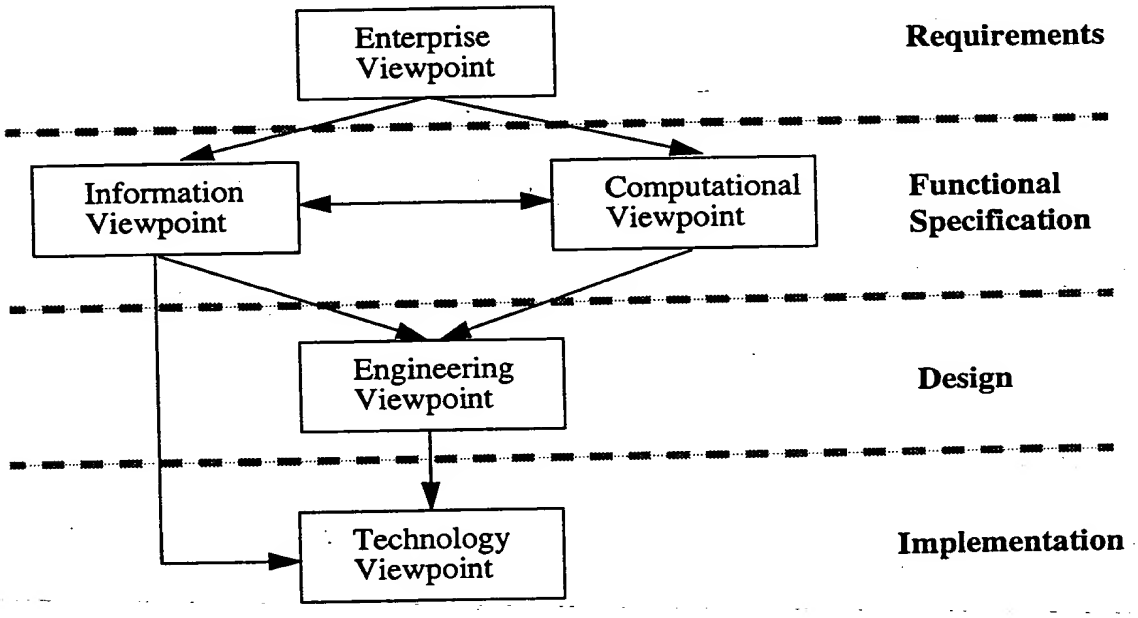


Figure 2.1 RM-ODP viewpoints and software engineering process

2.2.2 Transparencies

When contemplating a distributed system, a number of problems become apparent as a direct consequence of distribution:

- the system components are heterogeneous
- they can fail independently
- they are at different and possibly varying locations
- they need communications
- activities can be run concurrently
- it is difficult to satisfy real-time requirements

These problems can be solved either by the applications themselves or by standard solutions which provide transparency.

The transparency approach can lead directly to software reuse and alleviates the construction of applications. The designer expresses system requirements in form of a simplified statement of the interactions required and the transparency properties needed.

The transparencies defined in the RM-ODP are:

- **access transparency** masks differences in data representation and invocation mechanisms to enable interworking between objects. This transparency solves many of the problems of interworking between heterogeneous systems, and will generally be provided by default.
- **location transparency** masks the use of information about location in space when identifying and binding to interfaces. This transparency provides a logical view of naming, independent of actual physical location.
- **failure transparency** masks from an object the failure and possible recovery of other objects (or itself) to enable fault tolerance.
- **migration transparency** maskes from an object the ability of a system to change the location of that object.
- **relocation transparency** masks relocation of an interface from other interfaces bound to it.
- **replication transparency** masks the use of a group of mutually behaviourally compatible objects to support an interface.
- **persistence transparency** masks from an object the deactivation and reactivation of other objects (or itself).
- **transaction transparency** masks coordination of activities amongst a configuration of objects to achieve consistency.

2.3 TELECOMMUNICATIONS INFORMATION NETWORKING ARCHITECTURE (TINA)

2.3.1 The TINA Consortium

The Telecommunications Information Networking Architecture (TINA) Consortium is an international collaboration aiming at defining and validating an open architecture for telecommunications systems for the broadband, multi-media, and information era. The architecture is based on distributed computing, object orientation, and other concepts and standards from the telecommunications and computing industries [TIN95i].

The TINA architecture addresses the needs of traditional voice-based services, future interactive multi-media services, information services, and operations and manage-

ment type services, and will provide the flexibility to operate services over a wide variety of technologies. This vision implies a software architecture that offers reusable software components, supports network-wide software interoperability, eases service construction, testing, deployment and operation, and hides from the service designer the heterogeneity of the underlying technologies and the complexity introduced by distribution [TIN94c].

The intention is to make use of recent advances in distributed computing (e.g., Open Distributed Processing (ODP) and Distributed Communication Environment (DCE)), and in object-oriented analysis and design in order to improve interoperability, re-use of software and specifications, and flexible placement of software on computing platforms/nodes [RAC94]. In addition, the consistent application of software principles to both services and management software is a primary goal. The TINA architecture is furthermore ensuring that a multi-supplier/provider market for telecommunications services and management systems will be possible.

2.3.2 Decomposition of the architecture

The TINA architecture is decomposed into four main subsets [TIN95i]:

- The **service architecture** defines a set of concepts and principles for the design, specification, implementation, and management of telecommunication services [TIN96e].
- The **network architecture** defines a set of concepts and principles for the design, specification, implementation, and management of transport networks[1] [TIN95b]
- The **management architecture** defines a set of concepts and principles for the design, specification, and implementation of software systems that are used to manage services, resources, software, and underlying technology [TIN95g].
- The **computing architecture** defines a set of concepts and principles for designing and building distributed software and the software support environment. It also defines the distributed processing environment (DPE) that provides the support system allowing objects to locate and interact with each other. These concepts are based on the Reference Model for Open Distributed Processing (RM-ODP). The computing architecture refines, and adapts the RM-ODP standard, so that it is suitable for the design of telecommunication systems.

In addition to these subsets, the **overall architecture** contains the generic concepts and principles that should be applied to the design, specification and implementation of any type of software system in a TINA consistent way. In particular, the overall architecture contains separation principles that should be observed. The overall architecture is derived by extracting, and generalizing where appropriate, the common principles found in the four sets.

1. The term transport network encompasses both transmission and switching technology.

2.3.3 Relationships between the architectures

The overall architecture defines the common concepts and principles to be applied. The service, network, and management architectures must be consistent with the overall architecture.

The computing architecture must be consistently used within the service, network, and management architectures. This will ensure that software is constructed following the same basic principles, and will result in interoperable and portable software. The management architecture is specialized within the computing architecture for the purpose of managing the computing systems and software.

The service architecture requires service oriented abstractions of network resources. The network architecture provides these abstractions, providing services oriented views of connection establishment, modification, and release. The management architecture is specialised in the service architecture for the purpose of service management.

The network architecture provides service oriented abstraction of network resources that the service architecture relies upon. The management architecture is specialised in the network architecture for the purpose of network management.

The management architecture defines the generic concepts and principles suitable for the management of services, networks, and computing infrastructures. The management architecture must therefore be consistently applied, and specialised, within the service, network, and computing architectures. Management software, that has users interacting with it to perform management tasks, should be offered as services, and hence the concepts and principles of the service architecture can be applied.

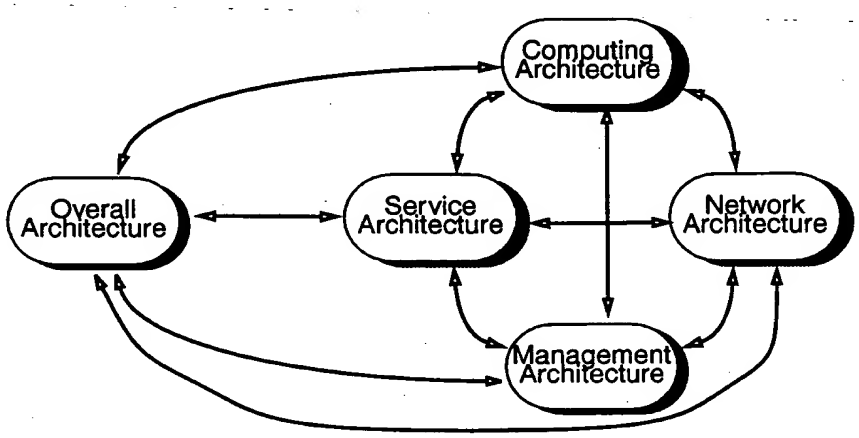


Figure 2.2 Relationships between architecture subsets

Figure 2.2 depicts the relationships between the architectures. Note that all the architectures are interrelated and that when one architecture relies on another, requirements are being made. These general requirement relationships are necessary to ensure that each architecture provides a suitable set of concepts and principles.

2.4 COMMON REQUEST BROKER ARCHITECTURE (CORBA)

2.4.1 The Object Management Group

The Object Management Group (OMG) is a consortium operated as a not-for-profit company based in the USA. The OMG's central mission is to establish an architecture and set of specifications, based on commercially available object technology, to enable **distributed integrated applications**. Primary goals are the *reusability*, *portability* and *interoperability* of object-based software components in distributed heterogeneous environments [Obja].

The **Object Management Architecture Guide** (OMA Guide) defines OMG's technical objectives and terminology, and provides the conceptual framework within which individual specifications are made. The guide includes a Reference Model which identifies and characterizes the components, interfaces, and protocols that compose the OMA [Objb].

Figure 2.3 shows the four major parts of the OMA Reference Model. The solid boxes represent software with application programming interfaces. The dotted boxes represent categories of objects with object interfaces.

- The ***Object Request Broker*** (ORB) enables objects to make and receive requests and responses transparently within a distributed environment.
- ***Object Services*** is a collection of services (interfaces and objects) that support basic functions for using and implementing objects.
- ***Common Facilities*** is a collection of services that provide general purpose capabilities useful in many applications.
- ***Application Objects*** are objects specific to particular end-user applications.

Through a series of RFIs (Request for interest) and RFPs (Request for proposal), OMG is populating the OMA Reference Model with detailed specifications for each component and service. The **OMG Object Model**, published in 1992, defines *common object semantics* for specifying the externally visible characteristics of objects in a standard implementation-independent way. The **Common Object Request Broker Architecture** (CORBA) specification, adopted in 1991, defines the programming interfaces to the ORB component.

An ORB provides the basic mechanism for transparently making requests to - and receiving responses from - objects located locally or remotely without the client needing to be aware of the mechanisms used to represent, communicate with, activate or store the objects. As such, it forms the *foundation* for building applications construct-

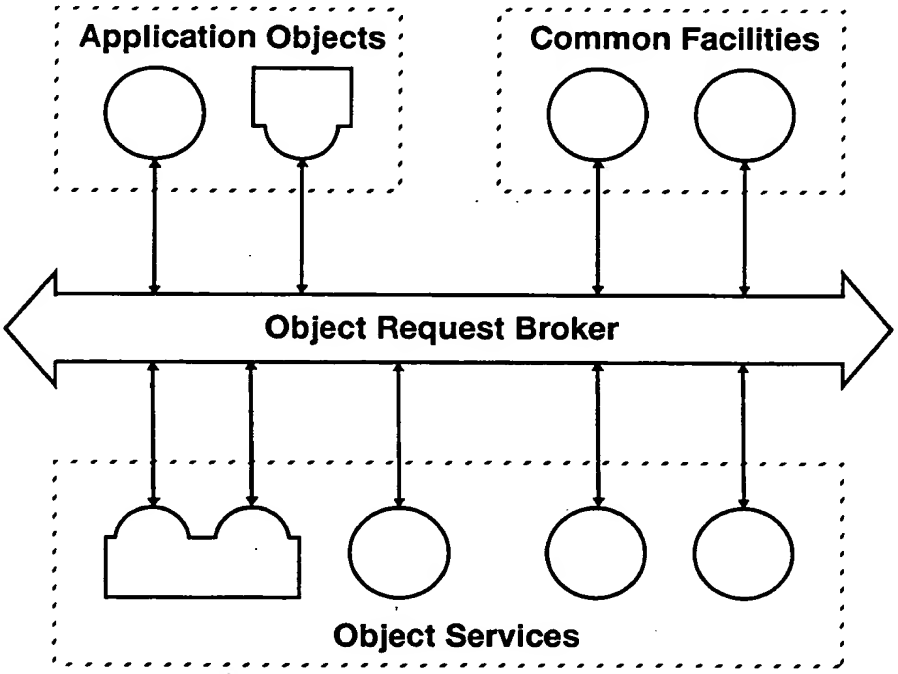


Figure 2.3 OMA Reference Model

ed from distributed objects and for interoperability between applications in homogeneous and heterogeneous environments. Objects made available through an ORB publish their interfaces using the Interface Definition Language (IDL) as defined in Chapter 4 of [Objа]. The IDL provides a programming language-independent way of specifying an object's operations and attributes.

2.4.2 The Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA) is structured to allow integration of a wide variety of object systems [BN95] [OHJ96] [OPR96].

Figure 2.4 shows a request being sent by a client to an object implementation. The Client is the entity that wishes to perform an operation on the object and the Object Implementation is the code and data that actually implements the object. The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect which is not reflected in the object' s interface.

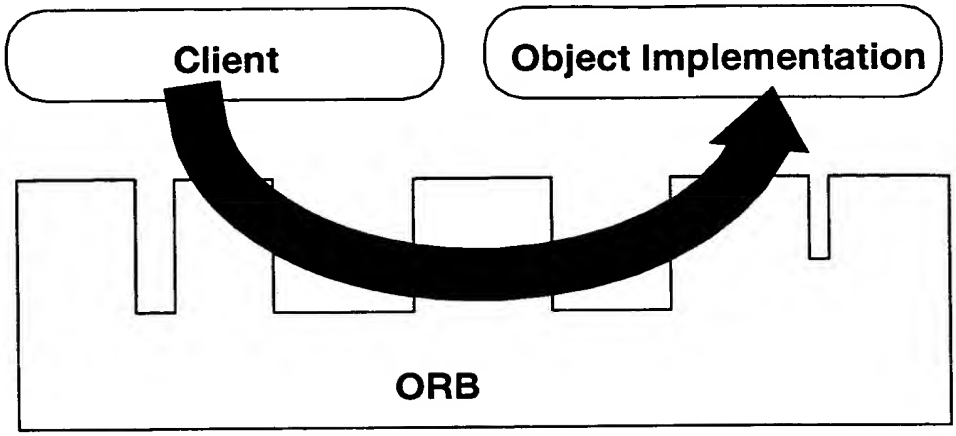


Figure 2.4 A Request Being Sent Through the Object Request Broker

Figure 2.5 shows the structure of an individual Object Request Broker (ORB). The interfaces to the ORB are shown by striped boxes, and the arrows indicate whether the ORB is called or performs an up-call across the interface.

To make a request, the Client can use the Dynamic Invocation interface (the same interface independent of the target object's interface) or an OMG IDL stub (the specific stub depending on the interface of the target object). The Client can also directly interact with the ORB for some functions.

The Object Implementation receives a request as an up-call either through the OMG IDL generated skeleton or through a dynamic skeleton. The Object Implementation may call the Object Adapter and the ORB while processing a request or at other times.

Definitions of the interfaces to objects can be defined in two ways. Interfaces can be defined statically in an interface definition language, called the OMG Interface Definition Language (OMG IDL). This language defines the types of objects according to the operations that may be performed on them and the parameters to those operations. Alternatively or in addition, interfaces can be added to an Interface Repository service; this service represents the components of an interface as objects, permitting runtime access to these components. In any ORB implementation, the Interface Definition Language and the Interface Repository have equivalent expressive power.

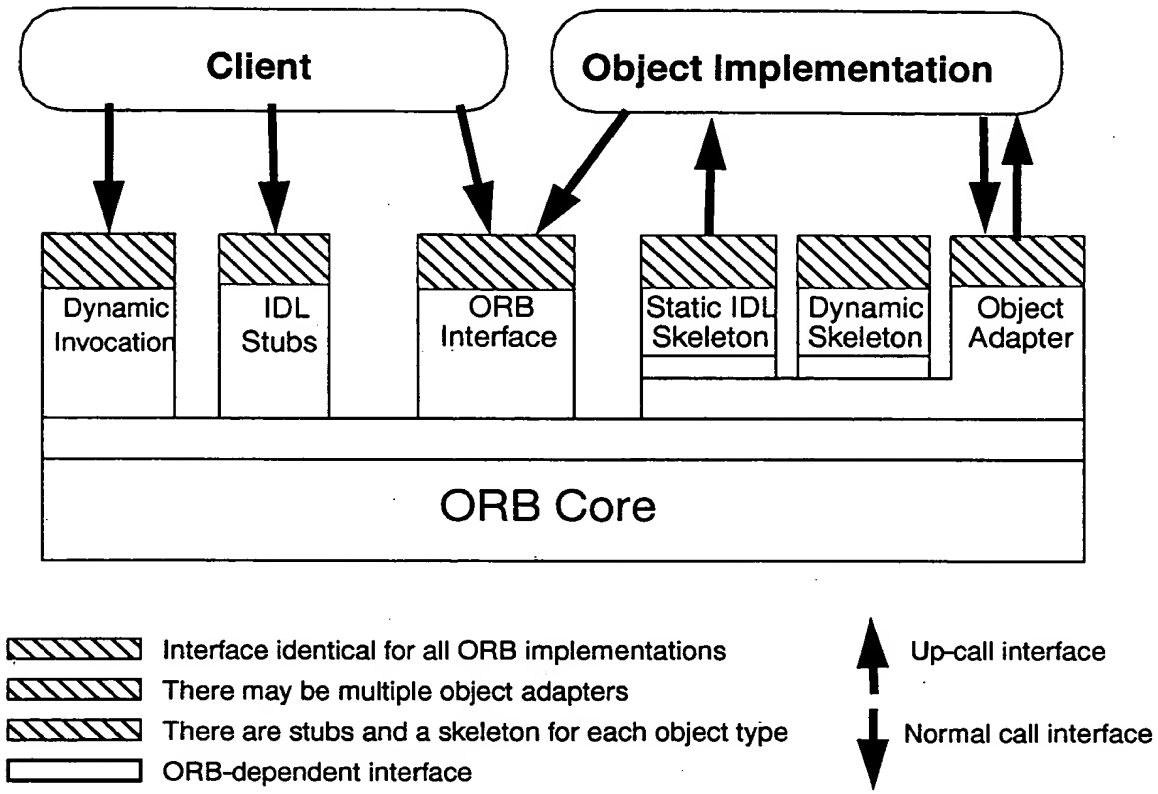


Figure 2.5 The Structure of Object Request Broker Interfaces

3

Related Work

3.1 INTRODUCTION

The Open Distributed Processing standardization aims to alleviate the design and programming of all types of distributed systems and, of course, also those with mobility characteristic. In ODP mobility is, however, considered as a specific characteristic of a type of distributed systems and is hence not studied in details in the ODP specifications [ITUa].

The TINA architecture is intended for the construction of telecommunications applications and has defined as one its objectives the support of mobility [TIN94c]:

The architecture should support (terminal, personal, session) mobility services.

However, TINA-C defines further [TIN96d]:

Definition of mobility concepts and associated generic information and computational object models are seen as a seamless part of the TINA-C architecture.

It is assumed that operational and stream interactions between user's objects running on a terminal and objects residing in the telecom system are always possible and nothing else but the default transparencies (access and location) are required. We will show in the next chapter that this is not true. The result is that there were no activity in TINA-C concerning mobility in 1994 when the work on this thesis was started. It was not until late 1995 that people began to realise that the support of mobility requires functionality in addition to the default transparencies. As a result TINA-C initiated activity in the Core Team to study mobility. In addition, some TINA auxiliary projects aiming to support mobility were initiated.

However, the approaches presented in these projects differ with the proposal presented in this thesis in some aspects:

1. Each project initiated by TINA aims to support only one type of mobility: either terminal mobility or personal mobility. Our proposal will support all types of mobility: terminal, personal and session mobility and in all flavours:

continuous or discrete. Our solution can be customized to fit all the telecommunications systems: private or public, wireline or wireless, wide-area or local area.

2. The support of mobility is integrated either in the DPE layer or in the Application/Service layer in the approaches initiated by TINA. In our approach, mobility functions are gathered into a functionally separated layer called the mobility layer. This layer is independent of both the DPE layer and the Application layer. As we will show in this thesis, the Generic Mobility System (GMS) realizing this mobility layer can be implemented as middleware, i.e off-the-shell software which can be installed in any open distributed system to provide mobility.

3. In TINA mobility-related issues such as security are only mentioned but almost not considered explicitly. The GMS contains also security functions which are necessary to protect the telecom system and the users against threats introduced by mobility.

4. In this thesis we also propose to consider mobility as an ODP transparency, i.e normal applications need not be aware of or interact directly with the GMS. Mobility transparency is not considered in any of these projects.

5. A service interface is also offered to the mobility-based applications (see Chapter 9.7) by the GMS. This kind of interface is not mentioned in any of the TINA projects.

6. Finally, our solutions for the support of terminal mobility and user mobility are different from the ones presented in the projects of TINA. We propose also a different system structure for the applications and a different approach to integrate them in the system.

This chapter introduces briefly the TINA activities concerning mobility. Three auxiliary projects which are related to mobility (PCS, DOLMEN and EURESCOM P608) are also presented.

3.2 TINA ACTIVITIES CONCERNING MOBILITY

3.2.1 User mobility and session mobility

In June 1996, TINA issued internally the document called *The Support of User mobility in the TINA Service Architecture* [Heg96] presenting a solution for user mobility. This document is intended to be included in the Service architecture document because user mobility should be inherently supported in TINA.

The document identifies three issues necessary for the support of user and session mobility:

- Registration
- Ubiquitous access and use of the system

- Security

A computational model for user and session mobility is also presented in the document. This model seems to be reasonable although completely different from the one proposed in this thesis.

3.2.2 Terminal mobility

In December 1996, an internal TINA-C report on Terminal mobility was released containing the results of a preliminary study about the impact of terminal mobility in TINA. The document studies three ways to support terminal mobility in a TINA environment:

1. Support of mobility functions in the Service Architecture
2. Support of mobility in the Resource Architecture
3. Support of mobility by the underlying infrastructure (kTN, TN)

All the three options have been evaluated and options 2 and 3 seem to be preferred because, according to the TINA Core Team, incorporation of terminal mobility functions should be avoided. Anyway, there is still no final conclusion and activities are still going on.

3.3 PCS

The ***Personal Communications Support in TINA*** is a TINA-C auxiliary project started in January 1996 in order to investigate the impact of PCS concepts on the TINA Service Architecture, in particular the TINA Access Session [TIN96c]. This project is performed jointly on behalf of Deutsche Telekom AG by the GMD Research Center for Open Communication Systems (FOKUS) and the Department of Open Communication System (OKS) at the Technical University of Berlin. The project is scheduled to be finished in June 1997.

The project focuses only on the first two aspects of the PCS concept, namely the support for personal mobility and the support of service personalization in TINA. The third aspect, i.e service interoperability within a TINA environment, is not addressed.

The project observes also that the "basic" TINA Access Session provides only some limited PCS capabilities and that an enhancement of the Access Session in respect to full PCS becomes necessary. The major target of the project is the definition of a "PCS-enhanced Access Session" which will be achieved by the identification, specification and implementation of new PCS-related Computational Objects and the enhancement of already defined computational objects related to the "basic" TINA Access session.

A prototype has been implemented and was presented at the TINA'96 conference in September 1996.

The computational model for user mobility support proposed by this project is extended from the TINA Service Architecture model and is different from the solution proposed in this thesis.

3.4 DOLMEN

The DOLMEN project - Service Machine Development for an Open Long-Term mobile and Fixed Networked Environment- is another TINA auxiliary project dealing with mobility [TIN96a].

The DOLMEN First Year Trial was executed in May 1996 (LAN tests) and in June/ July 1996 (mobile tests). The objectives of the trial were to establish the level of Quality-of-Service that a multimedia application obtains with current state-of-the-art mobile technology and to learn the limitations of that technology. The application used in the trial was Web information browsing. The early decision to use Web as the application was based on the facts that browsing is a good representative of applications expected to be used both in mobile and in fixed communication environments and that Web browsing is the most rapidly growing application used in any networked environment.

In the DOLMEN First Year Trial the end-users have laptops with PCMCIA-cards and GSM Phones. With these equipment the end-users have access to the GSM Data Service provided by the two national hosts involved in the DOLMEN project. By using the GSM Data Service the end-users obtain access to the services available in the fixed network.

The DOLMEN project has not only different objectives but also uses different concepts, platform and technologies than our work. It is presented here just for completeness since it is a TINA auxiliary project related to mobility.

3.5 EURESCOM P608

EURESCOM P608 - TINA Concepts for 3rd generation mobile systems- is a EURESCOM project started in August 1996 and has recently been approved as a TINA auxiliary project. The project is still in the starting phase and nothing much has been specified. It is worth mentioning that some results presented in this thesis are used as input to P608. The main objectives of this project are to:

- investigate how UMTS as defined by ETSI SMG5 can benefit from TINA principles in the areas of service and network management and administrative processes and to support the realization of UMTS standards in these areas based on TINA principles.
- specify how terminal mobility and handover can be taken into account in TINA specifications for DPE and connection management.

4

The Generic Mobility System

4.1 INTRODUCTION

The term mobility has become a vogue term in telecommunications and everyone has some opinion of what mobility is. But what exactly is mobility? Is it confined to mobile telephones and cellular networks? Is it a new kind of telecommunications service such as UPT (Universal Personal Telecommunication, cf. [CCI92c], [ITU94]) or PCS (Personal Communication Services)? Is it a new kind of telecommunications network such as GSM? The answers to these questions can ambiguously be both yes and no, depending on the viewpoint of the person answering the question and the context in which the question is presented.

In this thesis, mobility is viewed as an inherent capability of the telecommunications systems designed in accordance with ODP/TINA concepts [ITUa], [ITUb], [ITUc]. Mobility is thus not regarded as a telecommunication service but rather as an enhancement to the telecommunications systems. This enhancement does not come automatically as we will show: access and location transparencies are not sufficient alone to offer mobility support to the applications in a ODP/TINA telecommunication systems. In order to support mobility, a *Functional Separation Architecture* for the telecommunications system will be proposed. A functional layer will be dedicated to mobility functions and mechanisms. We will realize this mobility layer, by introducing a *Generic Mobility System.*

4.2 WHAT IS MOBILITY?

4.2.1 Definitions

In this thesis, a **telecommunications system** is defined as a system offering telecommunications services to the users who may be human beings, hardware devices or software processes. The telecommunications system consists of distributed hardware and software components which interact in order to provide services to the users.

This broad definition of the term **user** is important for several reasons. First, human beings, hardware devices and software processes will interact with the system in different ways. Second, they will require different types of platform support at the user interface, with different requirements for security, reliability, etc. Third and most important in our context, they may have different requirements for mobility.

We will use the term **subscriber** for the person or organisation who is responsible for the contract with the providers of telecommunications services.

In current networks, except wireless ones, the telecommunications services are delivered to a user at a fixed location which we will refer to as the **Network Access Point (NAP)** of the telecommunications system. The network access point names unambiguously the subscriber who is owning the contract for services at this network access point. In general, the NAP does not identify the user. The relationship between the subscriber and NAP is a one-to-many relationship since each subscriber may subscribe for more than one NAP.

In wireless networks (satellite, cellular, etc.) the concept of a NAP as described above is useless. A NAP in that interpretation may only exist as long as there is an active session (or call) between the user and the network. The NAP may then be identified in terms of frequency, time-slot or spectrum spreading code. However, in systems such as GSM, the NAP may even change during the session (e.g if the call is handed over from one base station to another).

In Chapter 5, we will redefine the NAP concept in order to make it useful also for mobile applications.

Mobility is an important aspect in telecommunications and the goal is to achieve mobility between networks of different kind and between accesses within the same networks. Universal Personal Telecommunications (UPT) is such an approach [CCI92c].

But even with UPT, it is difficult to achieve general mobility because the mobility component is embedded too deeply into the switching platform of existing networks. An in-depth analysis conducted by EURESCOM [EUR93] has shown that mobility across networks of different types (e.g. ATM, ISDN or cellular) and between networks owned by different operators is not possible unless radically new platform concepts are introduced. In current networks it is, for example, not possible to offer proper personal mobility in international VPNs.

There are several types of mobility which can be classified according to the category of user and the availability of services [Do96b].

a. Category of user

As mentioned earlier, a user can be a hardware device, a human being or a software process. This leads us to identify the following types of mobility:

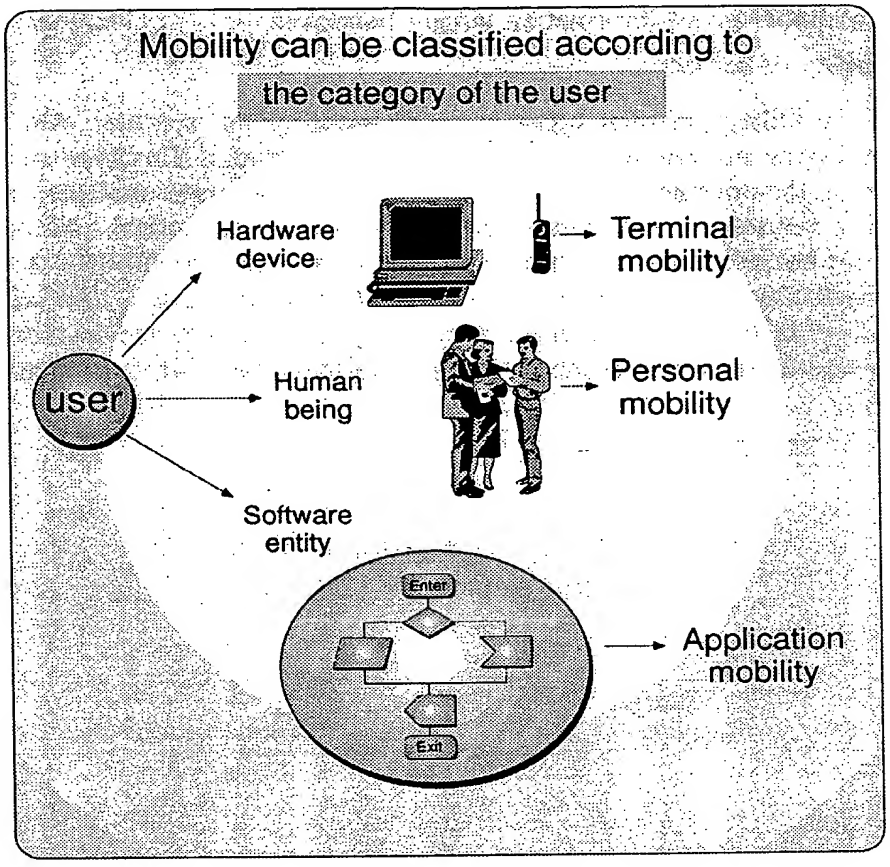


Figure 4.1 Mobility classified according to the category of users

Terminal mobility permits a terminal to change location while maintaining all the services. **Personal mobility** allows a human being to access or to be accessed by the network independently of where the access point and terminal used are located in the network and maintaining all services contained in the personal subscription. **Application mobility** allows a software process to be relocated from one machine to another or even moved between machines while processing.

Session mobility is defined as an added feature to those mentioned above. This mobility is ensuring that active sessions are not disrupted while terminals, persons or applications are moving or being relocated (However, sessions may be brought to a well-defined halt state in order to be resumed later). One example of session mobility is call transfer, e.g. a user can move from one terminal with multimedia presentation capabilities to one with voice-only capabilities or vice-versa. The presentation mechanism will change to meet the requirements of the new environment. Another example

is handover in cellular systems, permitting calls to continue while the mobile terminal moves from one cell to another. In GSM, the session is maintained without interruption, i.e without loss of information during handover.

b. Availability of services

Continuous mobility enables continuous availability of services while the user moves. Continuous mobility is offered in cellular networks. This type of mobility is making use of the mechanisms of session mobility. **Discrete mobility** enables the availability of services within certain areas and for certain access points, e.g home and office, but not while moving from one area to another. **Portability** is an example of discrete terminal mobility, where it is only allowed to move a terminal from one plug to another. In this case too, session mobility may be required in order to ensure that sessions are halted and maintained in well-defined states while the terminal is being relocated.

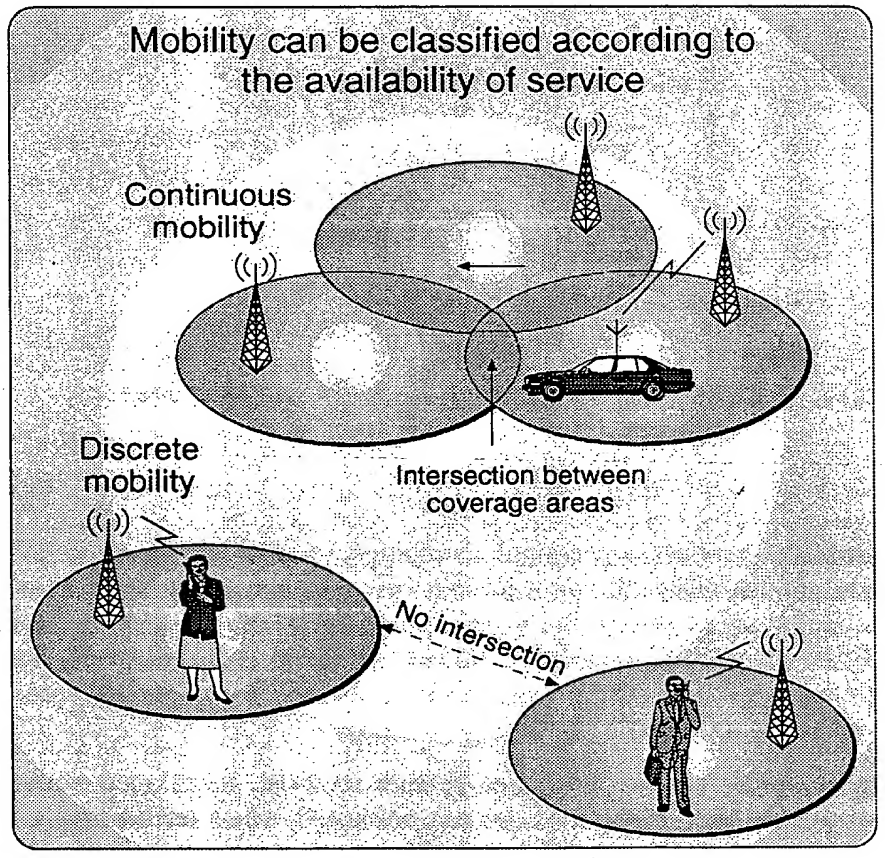


Figure 4.2 Mobility can be classified according to the availability of service

The distinction between continuous and discrete mobility has recently become less clear with cordless terminals (DECT or CTM (Cordless Terminal Mobility) [ETS95]) which are available continuously at some local areas but not necessarily between them. Applications such as intelligent agents is another example of partly continuous and partly discrete application mobility.

It is worth noting that mobility is not confined to any specific network. However, the types of mobility offered can be different. Continuous terminal mobility can only be supported in wireless networks while discrete terminal mobility, personal mobility and application mobility can exist in both wireless and wireline networks. Session mobility can exist in wireless networks, wireline networks and integrated wireless-wireline network.

Mobility is thus not a telecommunications service because mobility as such has no value for a user if not supplemented with other telecommunication features. Mobility enhances the availability of the telecommunications services.

4.2.2 How is Mobility currently implemented?

The mobility characteristic of telecommunications are currently realized by different mobile systems, each supporting one or two types of mobility. The systems are in general incompatible. GSM (Global System for Mobile Telecommunications) [GSM94c] in Europe and the Electronics Industry Association/Telecommunications Industry Association (EIA/TIA) Interim Standard 41 (IS-41) in North America support continuous terminal mobility. DECT (Digital European Cordless Telecommunications) [ETSb] [ETSc] and CTM (Cordless Terminal Mobility) [ETS95] allows limited continuous terminal mobility and discrete terminal mobility. UPT (Universal Personal Telecommunications) [CCI92c] [ITU:F.851] in Europe and PCS (Personal Communication Systems) in the USA are currently offering discrete personal mobility and will offer session mobility in the future. TETRA (Trans_European Trunked Radio Access) which is a a closed radio network, allows continuous terminal mobility.

UPT was originally intended to offer personal mobility between network access points in the fixed network and between networks of different types. In current realisations only the first objective has been met [AB93], while the second objective has been much more difficult to meet [EUR93]. Much effort is now put into obtaining mobility across fixed and mobile networks (this is one aspect of what is often referred to as fixed-mobile convergence).

The mobile systems are designed using different network architecture and different system structures. GSM has its own network architecture while UPT is based on IN (Intelligent Networks) architecture. Interoperability between systems is difficult and may lead to inconsistencies or errors. This phenomenon is also called feature interaction. The interfaces to the applications are also different making the introduction of mobility difficult and costly. The level of re-use is very limited. What is designed in GSM is not re-used in UPT. The only existing form of re-use is the knowledge and experience of designers participating in system specification and design.

4.2.3 Mobility and the applications

In this thesis, an application is defined as a distributed system consisting of a set of computational objects which interact to offer certain functions. An application is brought to the users as a service. In other words, a service is realized by one or several applications.

As the border between the telecommunication and computing worlds continue to blur, the applications increase dramatically both in number and in type. The applications are no longer confined to "telephony-based" applications such as freephone, premium rate, call forwarding, etc. but can be anything from a simple information service such as weather report to advanced interactive multi-media application.

Services can be classified according to their degree of mobility awareness as follows:

- Mobility-unaware applications
- Mobility-aware applications
- Mobility-based applications

4.2.3.1 Mobility-unaware applications

When mobility is introduced and supported in the telecommunications system, the existing "fixed" applications will become "mobile". The use of the term "mobile" is somewhat abusive and confusing here. The term "mobile" refers here to the mobility of the users, i.e the users can move while still having access to the applications. It emphasizes the availability of the applications to the mobile users but does not indicate anything about the mobility of the applications. The applications can migrate and be executed at a node nearby the position of the user but they can also be run at the home location of the user and presented at the current position of the user. With a Network Computer (NC) applications may be run in the network or in the terminal depending on the capabilities of the terminal, the cost of processing, time requirements, etc.

Ideally, it should not be necessary to modify or adapt these applications for them to become available to mobile users. They should be totally **mobility-unaware**. The designers of such application need not be aware of the functionality and underlying mechanisms necessary to support mobility. Examples of such applications are ordinary voice telephone, picturephone, video conference, information services, etc. They can also be familiar applications in the computing world such as word processor, database, spreadsheet, etc. More details about this type of applications will be given in Chapter 9.

4.2.3.2 Mobility-aware applications

In order to be able to support mobility the telecommunications system needs to acquire some mobility-related information such as user location, terminal capabilities, etc. This type of information can be collected directly by functions implemented tightly in the platform of telecommunication system or loosely by a new type of application called *registration applications*. By implementing the collection of data as an application, greater level of flexibility is achieved and it is also easier to customise the

registration according to the preference of the user. Such a registration application is **mobility-aware** since it collects mobility-related data. Another type of mobility-aware applications is called the *main application*. A main application is responsible for the support of session mobility, i.e offers to the user the possibility to suspend, move and resume a service session. More details will be given in Chapter 9.

4.2.3.3 Mobility-based applications

Mobility enables the emergence of a new class of applications, namely **mobility-based** applications, which uses the mobile characteristic of telecommunications. They can be general applications such as information services to mobile users. Examples of this type of applications are traffic information or weather reports which are filtered based upon the current position of the user. These applications can also be especially tailored for a special group of users such as taxi dispatch, mail tracking, point of sale, public safety, trucking, etc. This type of applications uses explicitly the mobility-related information possessed by the telecommunications system. For more details, see Chapter 9.

4.3 MOBILITY IN THE ENTERPRISE VIEWPOINT

Let us now start the study of mobility in the ODP/TINA context by the Enterprise viewpoint.

4.3.1 Basic concepts

As specified in [ITUa], the aim of an enterprise specification is to express the objectives and policy constraints on the system of interest. In order to do this the system is represented by one or more enterprise objects within a community of enterprise objects that represent the enterprise, and by the roles in which these objects are involved.

A **federation** is one particular kind of community; it is a coming-together of a number of groups answering to different authorities, i.e domains.

The **domain** concept is defined in the RM-ODP (Reference Model of ODP) [TIN95e], [ITUa] as follows:

"The domain concept allows the notations of autonomy, authority and control to be introduced in a distributed system. Objects of a RM-ODP domain are related because a certain part of their behaviour is controlled by the same authority".

Although defined in the Enterprise viewpoint, the domain concept is not confined to it but also appears and has impact on the information, computational and engineering viewpoints (This fact will be clearer as our modelling proceeds).

There are two general types of domains:

- An **administrative domain** covers a set of objects which are under authority

of an enterprise or organization. Administrative domain boundaries are also the points where changes of management responsibilities take place. Between administrative domains, either one or both sides may wish to impose their own access controls for such purposes as security, accounting and monitoring, in addition to controls imposed by the objects themselves.

- A **technology domain** is described to cover a set of objects which have identical representations and functionality of protocols, naming and addressing. Between technology domains protocol conversions and name translations are required.

Note that two administrative domains may belong to the same technology domain, and that an administrative domain may consist of several technology domains.

4.3.2 The TINA-C enterprise model

An Enterprise model presented by TINA-C [TIN96b] is as shown in Figure 4.3:

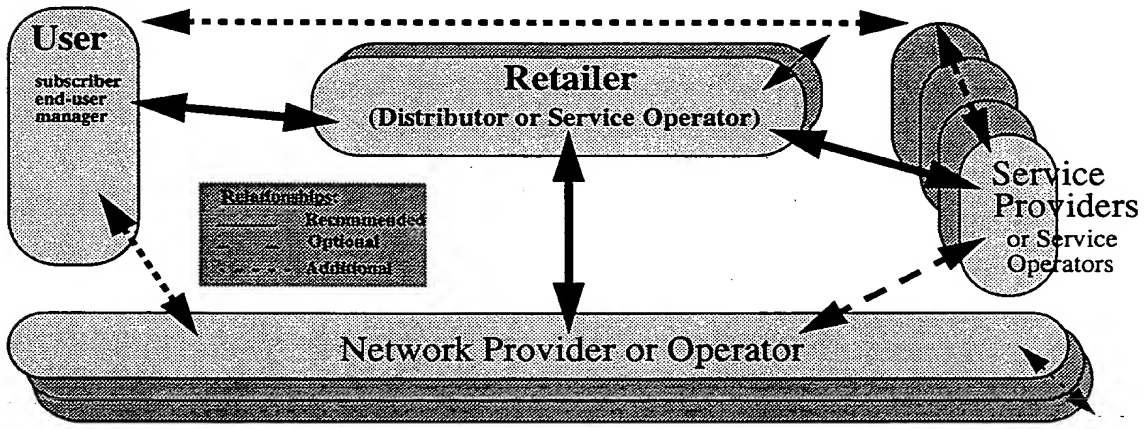


Figure 4.3 The TINA-C Enterprise model: An Information Services Supermarket Model

There are four separate domains depicted in the model; **User, Network Provider** (connection service provider), **Service Provider** (information service) and **Retailer** and two types of relationship between the domains, client-server (e.g. user-provider) and peer to peer (e.g. provider to provider).

The **user domain** represents the domain of interest of users of the services available in the retail domain.

The **network domain** represents the domain of interest of the owners of the network used to convey messages or data from the user to the retailer, service provider or other users.

The **service provider domain** represents the domain of interest of sources of the services which can be purchased in the retailer domain. It contains a set of disparate service providers, each of which provides agents which can negotiate with buyers in the retailer domain on supply and payment for services, and similar issues.

The **retailer domain** is perhaps the most interesting. It is intended to represent the domain of interest of a retailer who, amongst other things:

- Facilitates access to information and communication services and associated tools
- Acts as a middleman or broker for service providers and network providers
- Offers customized guaranteed services to individual customers
- Manages the services (a major point of added value)
- Provides a single point of contact for the user (customer, end-user, etc.).

This model is interesting in the sense that it presents a future view of telecommunications. Unfortunately, it does not focus on the mobility aspects and hence does not contribute much to the design of the functions and mechanisms to support mobility. Another enterprise model is needed.

4.3.3 Our enterprise model

Let us modify the TINA-C Enterprise model by focusing on mobility.

Taking mobility into consideration, the user will become the starting point of everything. The user wants to be able to access services while moving. In this context, which network provider he is using is not relevant to him. The **network provider domain** can therefore be "hidden" or associated with the **service provider domain**. Further, the user has only relation with the retailer who offers services to him. Which service provider this service originates from is not relevant to the user. The **service provider domain** can hence be unified with the **retailer domain**.

We define a **telecom system domain** which is a federation of one or more **retailer domains**, one or more **service provider domains** and one or more **network provider domains**.

Note that seen from one user, all other users that he interacts with can also be assimilated with the telecom system domain. This is a very important point since from one user's point of view it is not relevant where the other users are or how interactions can be realised. Therefore, seen from a user the telecom system domain may also comprise other user domains.

With personal mobility, a user must be able to use any terminal. Technologically, a terminal can be associated with the telecom system domain since it should be technologically compatible with the telecom system. However, a terminal belongs adminis-

tratively to a user since it is the user's property. On the other hand, it is neither convenient to associate a terminal to a user domain because the terminal currently used by a user may or may not belong to him. It is therefore more logical and convenient to allocate the terminal to a separate domain called **terminal domain**. What we have done is this

- maintained the TINA structure but separated the user domain and the network domain by a terminal domain
- hidden the details of the TINA model by use of federation since these details are not important for defining mobility support. These details are, of course important when defining services, but this is beyond the scope of this thesis.

Our enterprise model will therefore contain three domains: the **user domain**, the **terminal domain** and the **telecom system domain** as shown in Figure 4.4 where the telecom system domain is a federation of the domains defined by TINA, except the user domain. For the cardinalities on the relationships we have used the convention of OMT [RP91].

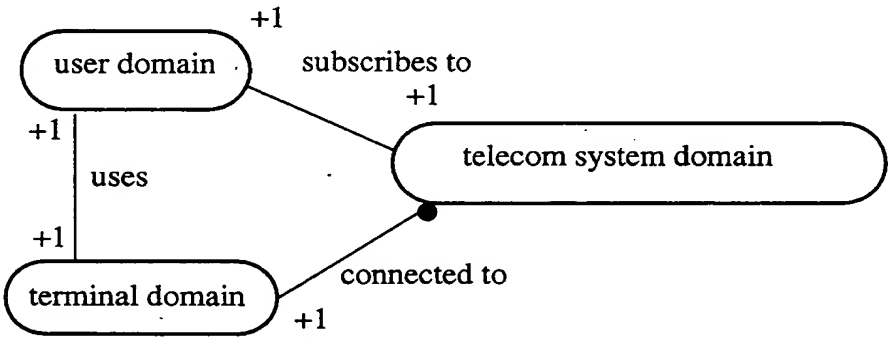


Figure 4.4 A simple Enterprise model with three domains

Further, we will require that a terminal cannot be operative unless a user is defined for it. The owner of a terminal may hence be defined as the default user of the terminal. The relationships between the domains are as follows:

A **user domain** subscribes to one or more **telecom system domains**. A **telecom system domain** may have relationships with one or several **user domains**.

A **user domain** may use one or more **terminal domains**. A **terminal domain** may serve one or more **user domains**.

A **terminal domain** is connected to zero or more **telecom system domains**. A **telecom system domain** may have connection with one or more **terminal domains**.

4.4 MOBILITY IN THE INFORMATION VIEWPOINT

4.4.1 Distinction between terminal, user and location

In traditional telecommunications networks such as PSTN (Public Switched Telephone Network), no distinction is made between the user or the terminal and the network access point (NAP), i.e. the point of attachment to the network. Calls and services can only be delivered to a NAP. A permanent association is made between the user or terminal to a location represented by the NAP.

If mobility is to be made available, then the mobile object, no matter whether it is a user or a terminal, cannot be permanently associated with a location.

In order to make terminal mobility possible, a distinction must be made between the terminal and the Network Access Point. A terminal must have an identity which is different of the one of the NAP and be recognisable by the network. The same applies for user mobility. The user's identity must be distinguishable from the identity of the terminal and be recognisable by the network. An initial information model for mobility contains the following objects (for more details about the information model, cf. [Aud96] [TIN95f]). Note again that the user may represent a person, machine or software process.

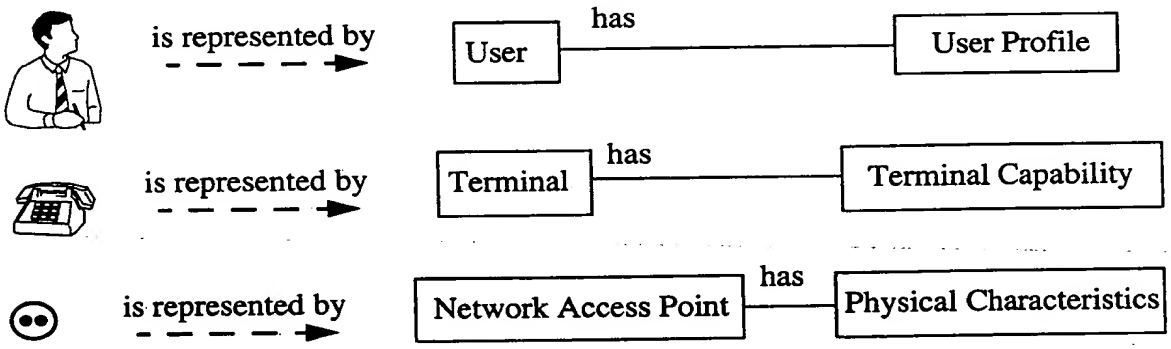


Figure 4.5 An initial information model for mobility

We have also shown that different characteristics can be associated with each of these objects. Each **Network Access Point** object has a **Physical Characteristics** object containing attributes such as type, protocol, bandwidth, etc. Each **Terminal** object has a **Terminal Capability** object containing attributes such as type, memory, resource, connection type, etc. Each **User** object has a **User Profile** object containing attributes such as service restriction, routing info, charging info. The **User Profile** object is important since customisation/personalisation of the services offered to the user requires that user profiles can be designed on an individual basis.

4.4.2 The relationships between User, Terminal and Network Access Point

To enable services to a user at a terminal connected to a NAP, a relationship, say **registered_at**, must exist between the **User** object and the **Terminal** object, and between the **Terminal** object and the **NAP** object as shown in Figure 4.6.

:

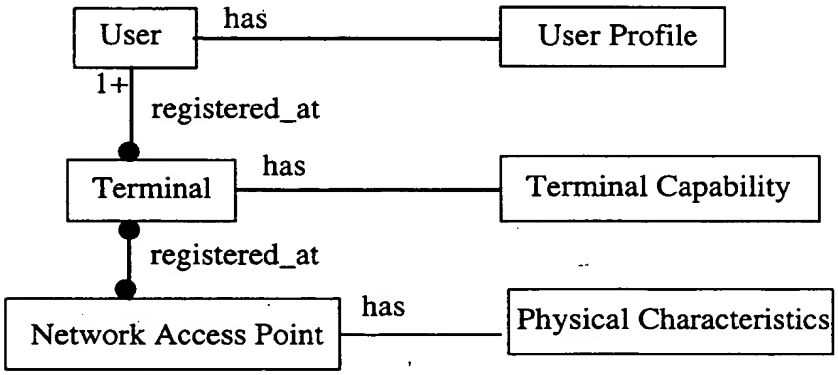


*Figure 4.6 The relationship **registered_at** between the User, Terminal and NAP*

A **Terminal** object can be **registered_at** zero or more **Network Access Point** objects. This covers the case of unconnected terminals and the case of advanced terminals having several connections to the network. On the other hand, zero or more **Terminal** objects can be **registered_at** one **Network access Point** object, i.e more than one terminal may be associated with one NAP (as for ISDN).

A **User** object can be **registered_at** zero or more **Terminal** objects. One or more **User** objects can be **registered_at** one **Terminal** object. This means that in our model a default **User** object is always associated with a **Terminal** object and that a **Terminal** object not associated with any **User** object has no access to the network.

Note, however, that the relationship **registered_at** does not alone ensure mobility. Mobility requires also some harmony and coordination between the data types **User profile**, **Terminal Capability** and **Physical Characteristics**. Let us take a simple example to illustrate this. A user wants to use some multi-media applications when it is possible. His preference is recorded in his **User Profile**. The user is now located at a place where he has access only to a simple terminal with only voice capability connected to a narrow bandwidth network access point. It is obviously not possible to run multi-media applications at this terminal. The best he can get is a voice-based application. The telecom system must somehow be able to coordinate all the three data types in order to find the best solution for the user. This coordination function will be studied later in this thesis (Chapter 9).

4.4.3 Mobility is ensured by the dynamic characteristic of the relationship

registered_at

If the relationship **registered_at** is static, i.e. defined once and for all at configuration time and having restricted modification possibility, then mobility cannot be offered.

Therefore, mobility relies totally on the dynamic characteristic of the relationship **registered_at**, i.e. its ability to change in concordance with the movement of the user or terminal. The degree of dynamics of the relationship **registered_at** is defined in terms of the real time correctness of the relationship **registered_at**. If the correctness can be ensured continuously, then continuous mobility is offered. On the other hand, if correctness can only be ensured discretely, then only discrete mobility can be offered.

Terminal mobility is ensured by the dynamic characteristic of the relationship **registered_at** between a **Terminal** and a **NAP**, i.e. by letting the terminal move from one NAP to another NAP.

User mobility is ensured by the dynamic characteristic of the relationship **registered_at** between a **User** and a **Terminal**, i.e. by letting the user change terminals.

The information model can only show the cardinality of the relationship **registered_at** and not its dynamic characteristic, i.e how it changes when the user or the terminal is moving. The dynamic characteristic can only be expressed in a computational model. The relationship **registered_at** is therefore reflected in a computational model by a set of operations towards the involved computational objects (e.g User_Agent, Terminal_Agent). Its dynamic characteristic relies then on the frequency of the execution of these operations and also on their characteristics such as response time, execution time, etc. This will be studied in detail in the following chapters.

4.5 HOW IS MOBILITY SUPPORTED IN ODP/TINA ENVIRONMENTS?

4.5.1 The TINA DPE architecture and distribution transparencies

According to the TINA computing architecture [TIN95a], a telecommunication application is realized by a set of interacting computational objects which rely on an abstract infrastructure called TINA Distributed Processing Environment (DPE) [TIN94a]. The DPE hides the complex details of mechanisms used to overcome problems caused by distribution.

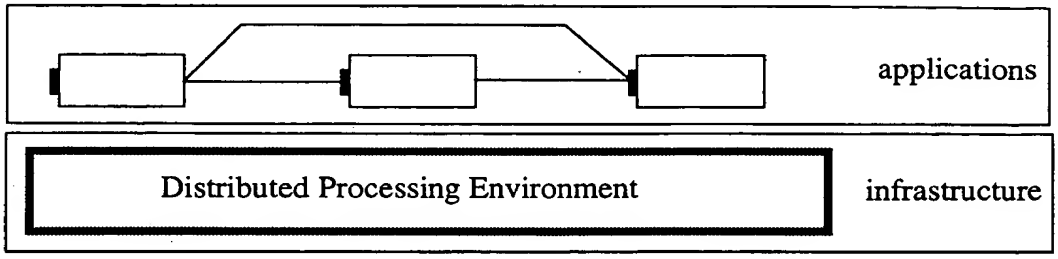


Figure 4.7 Abstract Infrastructure

The process of hiding the effect of distribution is known as distribution transparency in the Open Distributed Processing Reference Model (RM-ODP) [ITUa], [ITUb]. The application designers do not need to be aware of the mechanisms necessary to deal with different aspects of distribution and can therefore focus on their application specification. When addressing the distribution of their applications, they only have to express their requirements for transparencies. The properties of distribution is hidden or transparent to the end-users and the application designers in the enterprise, information and computational viewpoints.

The engineering viewpoint defines the mechanisms and functions by which those transparencies are realized. Each set of transparency properties requires the use of a set of standard functions in a specified way.

The distribution transparencies defined by TINA are:

- Access transparency
- Location transparency
- Federation transparency
- Migration transparency
- Transaction transparency
- Replication transparency
- Failure transparency
- Resource transparency
- Concurrency transparency

Physically, a DPE consists of several DPE nodes. A DPE node is a unit of resource administration providing support to the DPE architecture. The part of the DPE node supporting the DPE architecture is called a DPE platform. The computing resource supporting a DPE platform is called Native Computing and Communication Environ-

ment (NCCE). Even if the NCCE can itself be locally distributed, there is only one DPE platform associated with a DPE node [TIN94a].

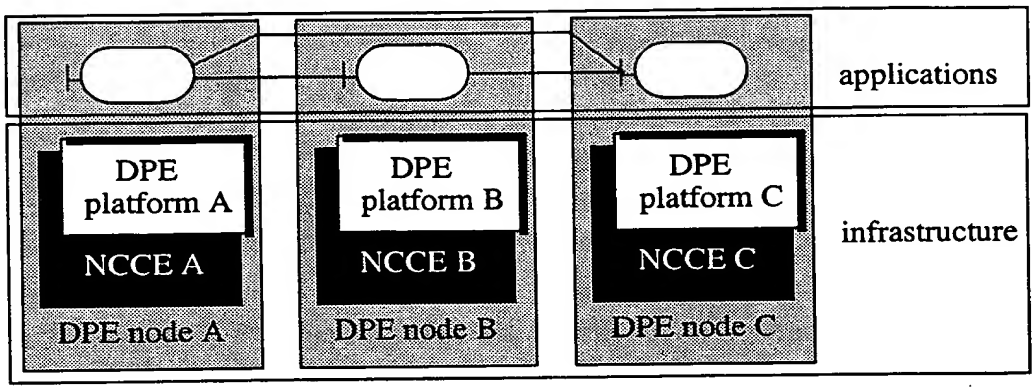


Figure 4.8 Physical Infrastructure

A TINA DPE platform shields the objects from the potential heterogeneity of the NCCE on which they are executed. Not all the transparencies are required to be provided by a DPE platform but only the ones called fundamental. Access transparency and location transparency are the two transparencies which are considered fundamental in the DPE architecture.

4.5.2 Mobility and the TINA Computing Architecture

Let us examine how mobility can be supported by the TINA Computing Architecture. In order to do so we shall explore how terminal mobility and personal mobility can be made available for telephony. Telephony is chosen as example for simplicity. Our discussion is, however, also valid for other network services. It applies also if we replace persons by software processes.

Consider three persons Tim, Carol and John. Tim has a fixed telephone subscription and a terminal with number 22 21 20 19. Carol subscribes to mobile telephony service and has the mobile telephone number 94 93 92 91. In GSM, this number identifies Carol's personal smart card and not the mobile station in which it is used. John has a Personal Mobility subscription (UPT or PCS) with personal user number 88 77 66 55. Tim can only be reached at home where his terminal is located. Carol can both make and receive calls anywhere she goes while carrying her terminal. John can register himself to make or receive calls (if authorised) at both Tim's and Carol's terminals or any other terminal.

Assume now that John registers himself at Carol's terminal. If Tim wants to call John, he dials John's personal number. The telecommunications system has to do two things before establishing the connection to Carol's terminal: first, find where John is

currently registered, namely at Carol's terminal, and, second, determine the current location of Carol's terminal.

4.5.3 An ideal model

Let us now model the situation described above in the computational viewpoint. We assume an ideal situation where each physical entity (persons, terminals, networks, etc.) can be represented as a computational object. A computational object may have both operational and stream interfaces. All the computational objects are supposed to rely on a TINA DPE supporting access and location transparencies

Each person is mapped to an instance of the object type ***user***, each terminal to an instance of the object type ***terminal***. The telephony service is mapped to an instance of the object type ***Telephony_Service***.

When ***user*** **Tim** wants to call ***user*** **John**, it makes a request to ***Telephony_Service***, providing ***user*** **John**'s number 88 77 66 55 as reference to the called object. Having the reference to ***user*** **John** and assuming access and location transparency, ***Telephony_Service*** can simply forward the request to ***user*** **John**. If ***user*** **John** is not busy and is willing to receive the call, then ***Telephony_Service*** can request the ***CSM*** (Communication Session Manager) [TIN95b] to establish a stream between ***user*** **John** and ***user*** **Tim**. The ***CSM*** may or may not interact directly with the objects ***user*** **John** and ***user*** **Tim** to establish a stream; this is indicated with dotted lines in the figure. The stream will naturally go through the terminals and the whole network before ending at the users, but at this level of abstraction where distribution is hidden, a stream can be logically regarded as established directly between the users.

In such an ideal model, the object ***user*** **John** can move anywhere and still receive telephone calls. In order to reach the object ***user*** **John**, any other object just needs its reference or number. The data format used when communicating and the location of object ***user*** **John** is abstracted away by the access and location transparencies offered by the DPE. Personal mobility is therefore ensured in the ideal model by the DPE.

The same reasoning can be applied to terminal mobility. A stream can always be established between two terminal objects independently of their locations, again assuming the access and location transparencies of by the DPE.

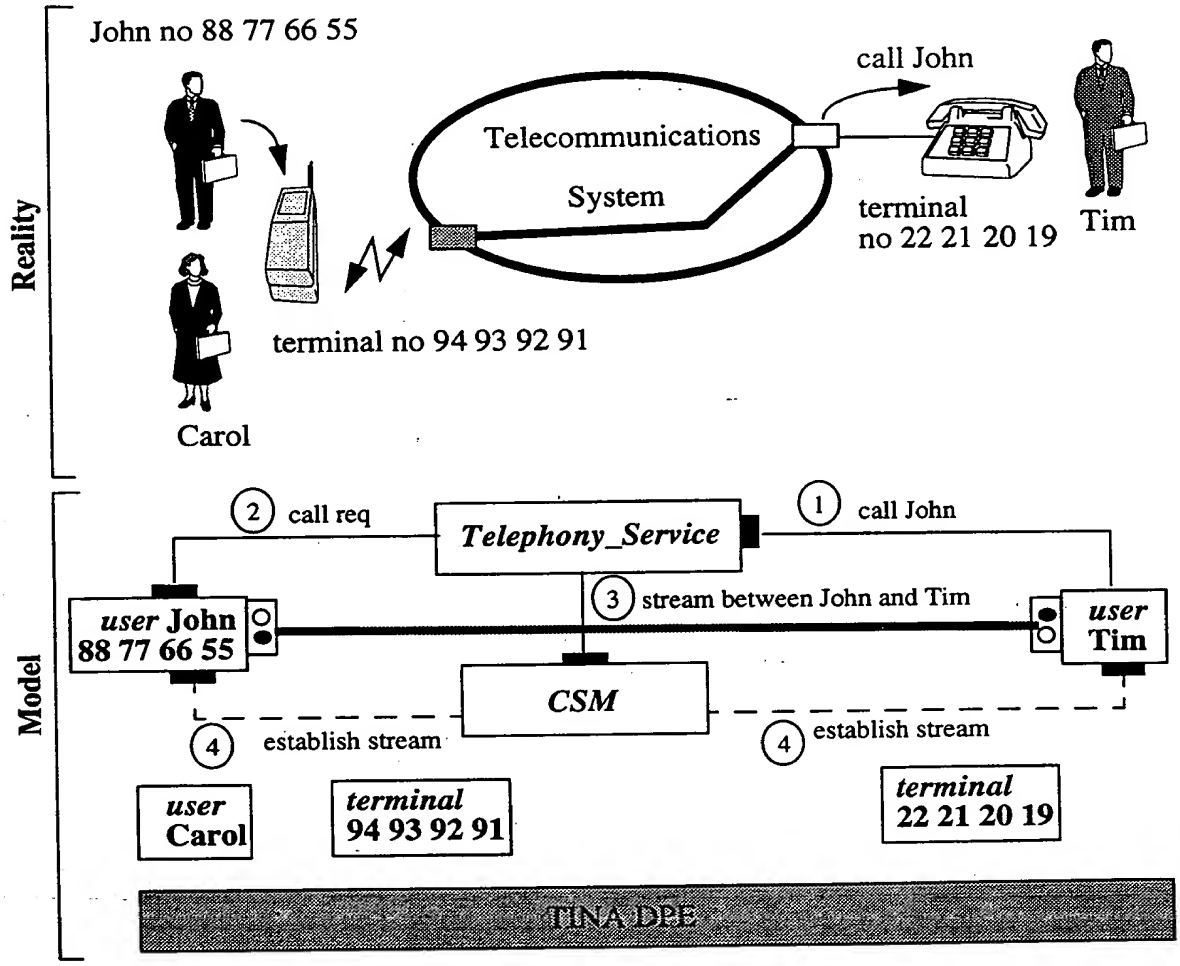


Figure 4.9 An ideal model of a telecommunications system supporting mobility

We may therefore conclude:

In an ideal model where all objects are accommodated on a DPE, personal mobility and terminal mobility are seamlessly supported provided that access and location transparencies are in place.

4.5.4 A real model

Unfortunately, the model of Paragraph 4.5.3 is naive. In fact, we cannot in general assume that a user is accommodated on a DPE platform. This applies even if the user is a software process since we cannot require that communicating software processes

are located at a machine with DPE. A user may therefore in general be considered as a non-DPE node. In addition, the user is not part of the telecommunications system.

Therefore, the object ***user*** will neither belong to the same administrative domain nor to the same technological domain as the telecommunications system.

In our example, we may assume that each object ***user*** constitutes its own administrative and technology domain. Our system may be partitioned into four technology and administrative domains as follows:

- Domain "***user John***" consists of object ***user*** **John**
- Domain "***user Carol***" consists of object ***user*** **Carol**
- Domain "***user Tim***" consists of object ***user*** **Tim**
- Domain "***telecom system***" consists of objects ***Telephone_Service***, ***CSM*** and ***terminal***s

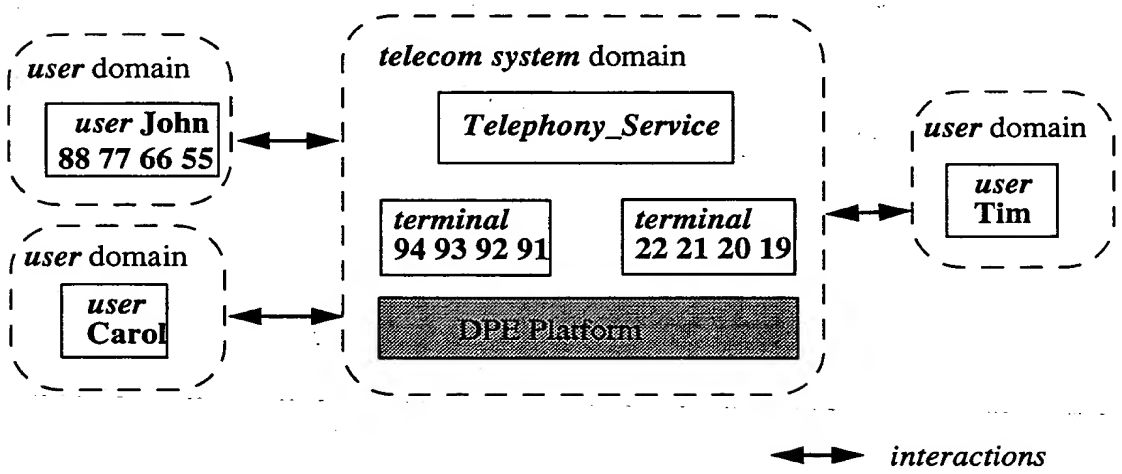


Figure 4.10 The model partitioned into domains

Note already at this point that the ***telecom system*** domain may be a composition of several interacting administrative and technology domains. This has no impact on the modelling of mobility.

Let us now examine how the domains of Figure 4.10 interact and how mobility is supported in this new model.

ODP [ITUa] defines a notion of interceptor which stands at the border between two domains and enables or permits interactions between them. Interceptors are of three types:

- "**Gateways**" i.e objects connecting two networks and performing data conversion.
- "**Agents**" i.e objects acting on behalf of other objects or entities which are not objects in the sense of ODP
- "**Monitors**" objects which are responsible for surveillance of other objects or entities.

Protocol conversion and name translation are required between technology domains and are supported by interceptors. Such interceptors are referred to as **In-line interceptors** (Figure 4.11).

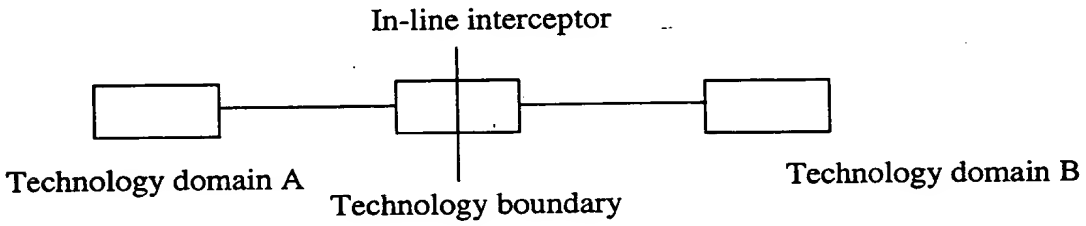


Figure 4.11 In-line Interceptor - Technology boundary

Protection and security functions are required between administrative domains. These functions are supported by **Split interceptors**. These interceptors are active only during the security checking and other administrative exchange of information between domains.

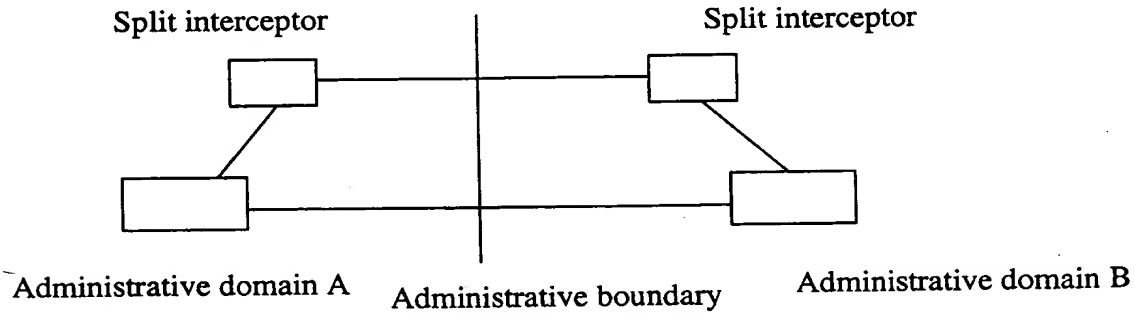


Figure 4.12 Split Interceptor - Administrative boundary

Where an administrative and a technology boundary co-exist, the two types of interceptors can be **combined** (Figure 4.13).

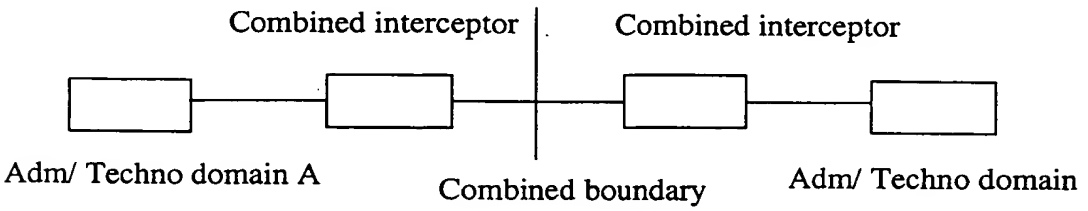


Figure 4.13 Combined Interceptor - Combined boundary

In our example, we assume that there is a technology boundary between the ***user*** domain and the ***telecom system*** domain. This boundary is also referred as a perceptual reference point [ITUc], [TIN95e] representing a point between the system and its environment (e.g the man-machine interface). All the interactions at this reference point go through an object ***terminal*** which acts as an in-line interceptor. It translates information represented in a machine format into another format which is comprehensible for the ***user*** object.

Between the domains ***user*** **John** and ***telecom system*** there is in addition an administrative boundary because the ***telecom system*** must be able to identify and authenticate the ***user*** **John** before allowing him to access and use the services (this requirement is similar to the one of UPT). In the domain ***telecom system*** we introduce therefore an interceptor, ***PD_User_Agent*** (Provider Domain User Agent), which represents the object ***user*** **John**. This interceptor is responsible for the protection and security functions. The TINA Service Architecture [TINA:servic_arch] also adopts a similar solution and defines a User_Agent for each user. Note again that user may be a person, machine or software process in order to avoid interpreting the model in a too narrow sense.

Let us return to our example where object ***user*** **Tim** wants to call object ***user*** **John**. Object ***user*** **Tim** communicates via a man-machine interface offered by object ***terminal*** **22 21 20 19** and issues a `Call_John` request at that interface. The object ***terminal*** **22 21 20 19** forwards the call request to object ***Telephony_Service*** which requests the ***PD_User_Agent*** **John** to provide the address to which the call is to be routed. Having received this information, the ***Telephony_Service*** object will then request the ***CSM*** object to establish a stream between ***terminal*** **22 21 20 19** and ***terminal*** **94 93 92 91**.

If the object ***PD_User_Agent*** **John** does not know the current location of the object ***user*** **John**, i.e the terminal number where John is currently registered, then it will not be possible to call ***user*** **John**.

It is observed that personal mobility is no longer automatically supported in the Telecom_System although it has a DPE supporting access and location transparencies.

This is so because the user objects are no longer accommodated on the DPE but represent objects which are technologically and administratively separated from the platform. Note that this is so even if the user objects represent software processes or machines.

In order to support personal mobility the following functionality must be added to the ***PD_User_Agent***:

- Function to store user location information
- Function to update user location information

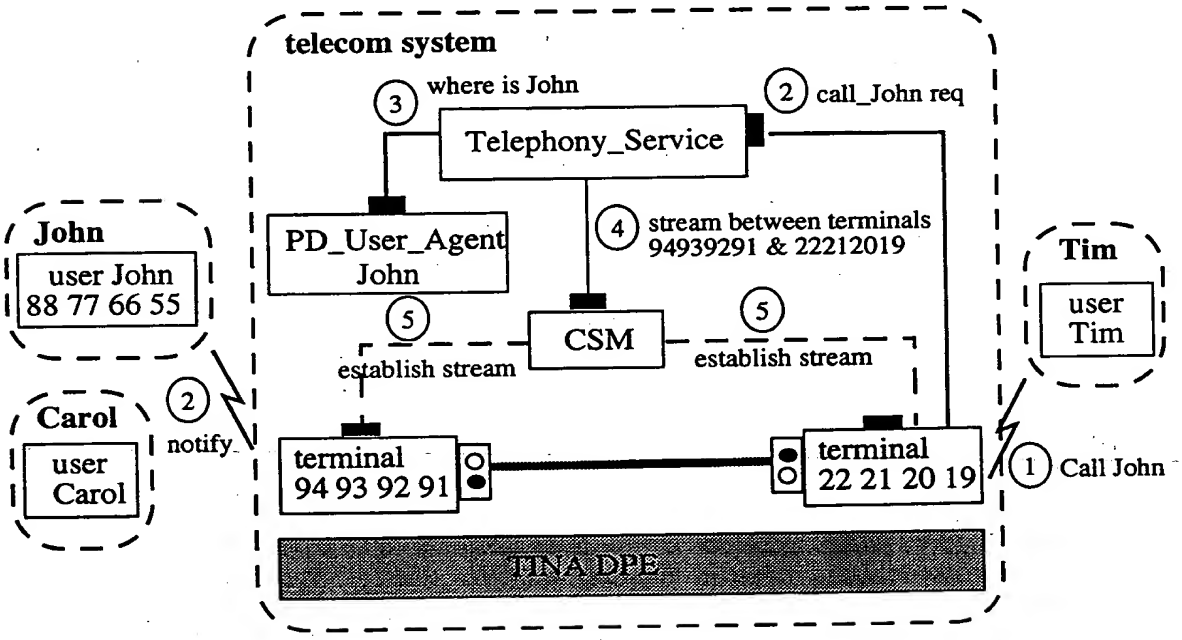


Figure 4.14 A model with users as separate domains

It is worth noting that the location information is subject to frequent change due to the mobility of the user. The ***PD_User_Agent*** object must have an operational interface allowing the ***user*** object to register its location. Of course, this interface is not direct since communications between the ***user*** object and the ***PD_User_Agent*** object always go through a terminal.

Another difficulty arises at the Engineering viewpoint where distribution is taken into account. The objects ***user***, ***PD_User_Agent*** and ***Telephony_Service*** may be geographically located far away from one another. In order to fulfil real-time requirements, it may be advantageous to elaborate strategies for replication and migration of the ***PD_User_Agent*** which optimise the location updating and location retrieval algorithm.

4.5.5 More improvement of the model

So far all the ***terminal*** objects are considered as belonging to the domain telecom system. Technologically such partitioning can be justified because the terminal objects may reside on a DPE platform and compatible with all other objects such that no protocol conversion and name translation are needed. However, in order to be general we may as well assume that the terminal domain and telecom system domain also belong to different technology domains or at least requiring name translation. This applies even though the terminal contains a DPE.

Administratively, a terminal object may thus belong to a separate domain even though it contains a DPE. Of course, there are terminals which are not TINA DPE nodes; however, they need to be treated in the same way as we treated the ***user*** object above.

In Figure 4.15, we have introduced two new domains **terminal** representing the ***terminal*** objects. Between two administrative domains interceptors are required. In the domain **Telecom_System** a ***PD_Terminal_Agent*** object representing the object ***terminal*** is introduced for supporting security functions, i.e identification and authentication of the terminal, combined with name translation. This object is thus a combined interceptor between the two domains. However, the ***PD_Terminal_Agent*** object of the object ***terminal 22 21 20 19*** may have minimum or no security functionality because the terminal is attached to a fixed access point and the risk for fraudulent use is minimal. This is in fact how telephony service is currently implemented.

In the domain **terminal**, a ***Service_Provider_Agent*** object may also be required to protect the terminal against "pirate" communications systems which want to extract confidential information from the terminal for later fraudulent uses. Such ***Service_Provider_Agent*** objects do not exist in most current digital cellular telephone systems. However, DECT has functional support for them.

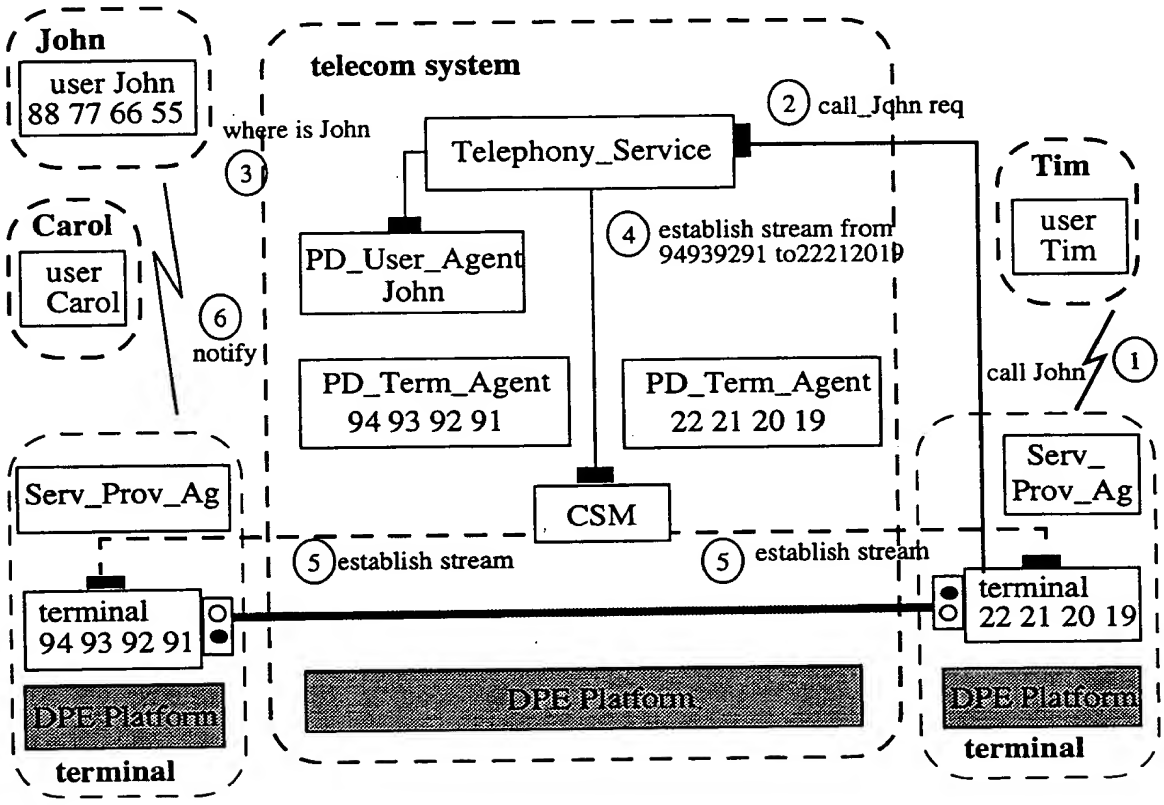


Figure 4.15 A model with terminals as separate domains

Let us now examine how terminal mobility can be supported in this new model. Each domain **terminal** may still be supposed to reside on a DPE platform supporting access and location transparencies. However, there may not be continuous operability between the terminal DPEs and the DPE of the telecommunications system. We shall now see what consequences this will have on the model

If the DPE platforms are able to interoperate to support access and location transparencies, then, theoretically, all the objects in the domain **terminal** can interact directly with any object in the domain **telecom system** and other **terminal domains**. The ***CSM***, on request, can establish a stream between two arbitrary terminals or between a terminal and an arbitrary object.

Unfortunately, interoperability between the **terminal** DPE and the **telecom system** DPE does not always exist. The mobile terminal is a particular DPE node with special behaviour:

- It changes frequently the node with which it has direct link.
- It may just disappear from a node and reappear later at any other node.

We may then conclude that terminal mobility is no longer automatically supported since interoperability between the DPE platforms is not guaranteed at all time.

In summary, we may conclude that the access and location transparencies defined for ODP and TINA DPEs are not sufficient for supporting terminal and user mobility. [Do96c]

The added DPE functions required to support terminal and personal mobility will be studied in more details in the following chapters.

4.6 A FUNCTIONAL SEPARATION ARCHITECTURE SUPPORTING MOBILITY

As shown in the previous paragraph, the access and location transparencies defined for ODP and TINA DPEs are not sufficient to support mobility. In order to support mobility we propose the Functional Separation Architecture shown in Figure 4.16:

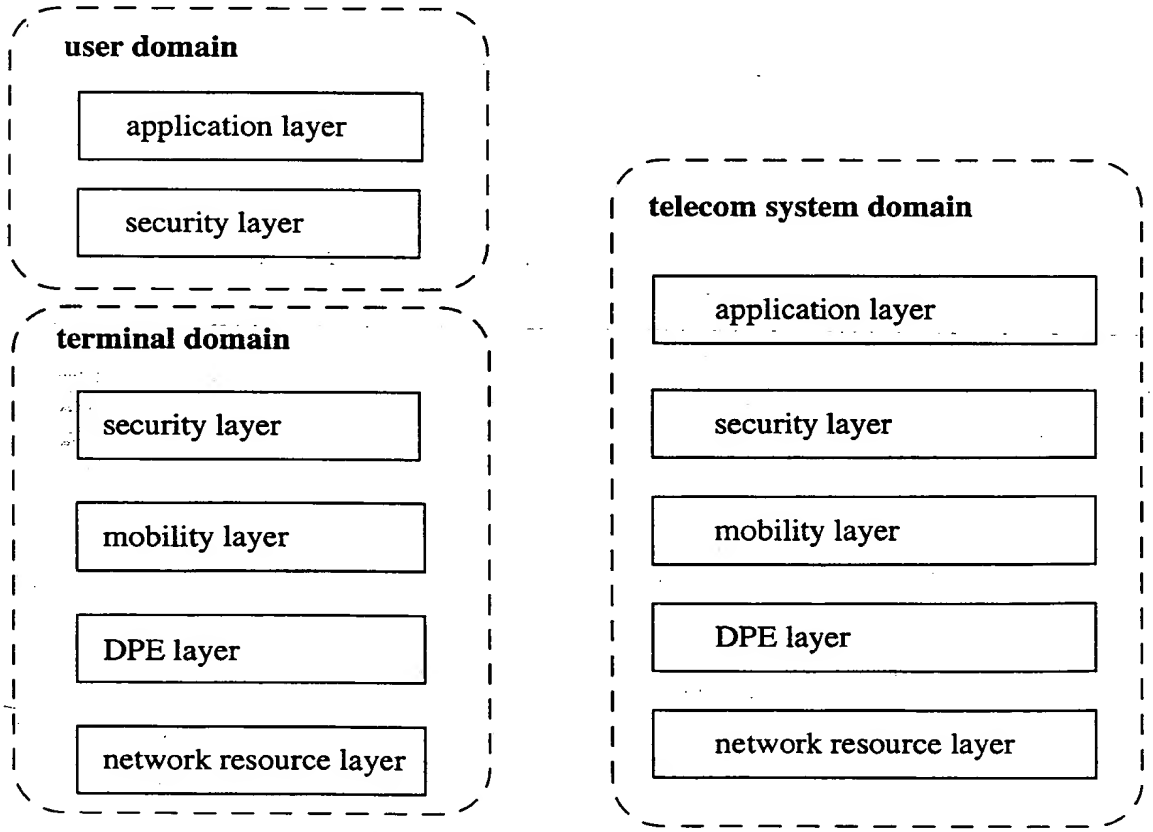


Figure 4.16 The Functional Separation Architecture supporting mobility

The whole system consisting of users, terminals and telecommunications system is, as explained in Paragraph 4.3.3, represented by a model consisting of three domains: **user** domain, **terminal** domain and **telecom system** domain.

In each domain, functions are grouped according to their nature into functional layers. These layers do not imply any hierarchical relationship but only a functional separation. An object in any layer can interact directly with any object in any other layer via offered interfaces. The same functional separation concept is used in the INA Framework Architecture [INA93], the OSCA Architecture [Bel92] and the TINA-C architecture [TIN95e].

In the **telecom system** domain, we have defined five functional layers: Network resource layer, DPE layer, Mobility layer, Security layer and Application layer.

- The **network resource layer** comprises the objects controlling/managing the transport network which is used in the establishment and release of stream flows.
- The **DPE layer** comprises objects of the distributed computing infrastructure which implement the distribution transparencies. The DPE layer provides various DPE services to the applications.
- The **mobility layer** comprises the objects which enable the support of mobility.
- The **security layer** comprises objects which are dealing with security issues.
- The **application layer** comprises the objects which make use of the functions offered by other layers to provide services to users.

The objects in the different layers communicate internally in the layer as well as with objects in the other layers. The distinction between the application layer and the network resource layer reflects the call and connection separation principle introduced by the IN (Intelligent Network) architecture [Aud92], [Sei92]. Objects in the network resource layer use services from the DPE layer. Reciprocally, the DPE layer can optionally use the network resource layer to establish a bearer for the kernel transport network (kTN). The mobility layer uses services from the DPE layer, the network resource layer and the security layer. The security layer uses services from the DPE layer and the network resource layer. Finally, the application layer may use all other layers to realize its functions.

It is worth noting that TINA-C defines only three layers: network resource layer, DPE layer and service layer [TIN95e]. Indeed, during the time this thesis was elaborated, TINA-C has only a few activities concerning mobility and security (see Chapter 3). The architecture presented here is also modified and improved compared to the one presented in [Do96b].

In the **terminal** domain, there are four functional layers corresponding to the ones in the telecom system domain: network resource layer, DPE layer, mobility layer and security layer. The reason that there is no application layer in the **terminal** domain is because we choose to associate the usage of a terminal with a user. We shall see in

Chapter 6 that it is convenient to demand that a terminal is always associated with a default user who may be the owner of the terminal.

In the **user** domain, there are two functional layers: security layer and application layer. The security layer comprises objects which assist the user in the security procedure (See chapter 7).

It is worth noting that in order to communicate with the **telecom system** domain, the **user** domain always requires a **terminal** domain. In fact, the domain partitioning here is an administrative one which does not correspond to a technological one. Technologically, the two domains may be unified onto one domain. However, the analysis is independent of whether or not these domains are separate technology domains. Application objects of the **user** domain are running on the **terminal** domain. The relationship between the **user** domain layers and the **terminal** domain layers is therefore similar to the relationship between layers of the same domain. Objects in the application layer of the **user** domain may use all the layers of the **terminal** domain.

The relationship between the **user** domain layers and the **telecom system** domain layers and the relationship between the **terminal** domain layers and the **telecom system** domain layers are of peer-to-peer type, e.g **terminal** domain network resource layer may communicate **telecom system** domain network resource layer, etc.

The proposed Functional Separation Architecture yields a high level of modularity and flexibility. The layers are autonomous and can be designed and implemented separately. Any internal changes in one layer does not affect the other layers as long as its interfaces remain unchanged.

One layer that is special and deserves to be mentioned separately is the DPE layer. The DPE layer implements the mechanisms necessary to deal with the different aspects of distribution and supports therefore the distribution transparencies. So ideally, the functionality offered by the DPE layer, although used by the application layer, should be invisible to it.

This observation will play an essential role in our design of the mobility layer. Although providing mobility support to the application layer, the mobility layer should be designed in such a way that it is transparent to the applications which are not using mobility as an integral part of the application (see Paragraph 4.2.3). The definition of this transparency is the purpose of this thesis.

4.7 THE GENERIC MOBILITY SYSTEM AND MOBILITY TRANSPARENCY

4.7.1 Objective

In the previous paragraph, we have presented a system architecture supporting mobility. In this paragraph, we proceed with the construction of the mobility layer. In order to realize the mobility layer, a **Generic Mobility System (GMS)** is proposed.

The GMS encapsulates all the functions needed to support mobility, i.e all the processing and data management necessary to support the mobility of the users and terminals are taken care of by the GMS.

The mobility-unaware applications (Paragraph 4.2.3) which constitute the majority of telecommunications services, shall not even need to be aware that the users and the terminals are moving. They do not need to be aware that the GMS is used in order to offer services to mobile users. Mobility is hence made transparent to them by the GMS. In the terminology of ODP, mobility is thus regarded as an additional transparency to the already-defined distribution transparencies which are defined by ODP and TINA [Do96b].

The objective of GMS is thus to support mobility transparency in an ODP/TINA system.

On the other hand, the mobility-based applications (Paragraph 4.2.3) need functionality related to mobility and may explicitly request GMS to perform specific tasks.

The GMS is generic in the sense that it is applicable to all the telecommunications systems. A telecommunications system, public or private, can choose, adopt, customize and instantiate every component of the GMS to obtain a mobility system supporting the type of mobility required. One instance of the GMS may for instance support only discrete terminal mobility suitable for a DECT system while another instance may support all types of mobility, i.e terminal, user and session mobility. The service applications of the telecommunications system can then in turn choose and subscribe to the services of the mobility system.

The GMS can be implemented as a middleware, i.e off-the-shelf software which can be purchased and installed in any ODP system to support mobility. The flavour of the wanted mobility can be combined and configured at installation.

4.7.2 Structure and composition of the GMS

Since there is a mobility layer both in the **terminal** domain and the **telecom system** domain, the GMS shall span both domains as shown in Figure 4.17.

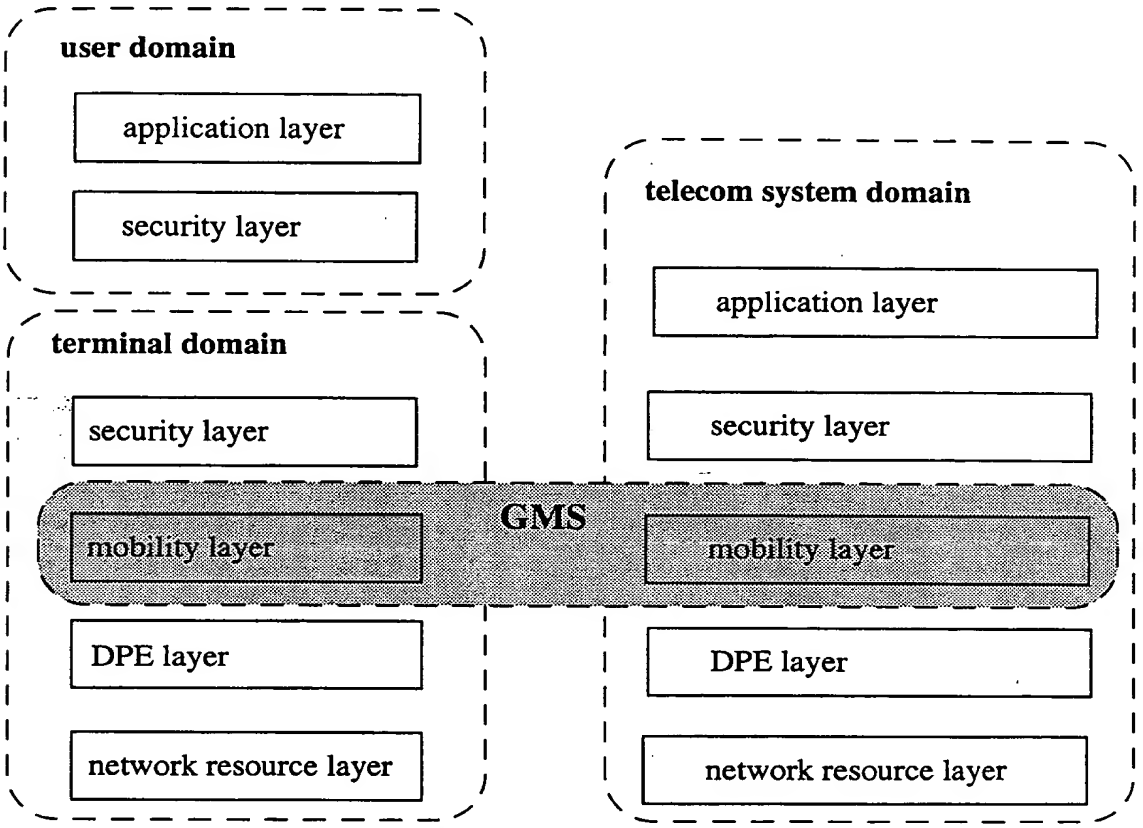


Figure 4.17 The Generic Mobility System (GMS)

The Generic Mobility System must have the following fundamental functions:

- **Terminal Mobility** comprising functions and mechanisms to support terminal mobility at object communication level.
- **User Mobility** comprising functions and mechanisms to support user mobility at object communication level.
- **Mobility Transparency Support** comprising functions and mechanisms to make mobility transparent at object communication level.
- **Mobility-related Security** comprising functions and mechanisms to handle the mobility-related security issues.
- **Mobility-unaware Application Support** comprising functions and mechanisms to support and make mobility transparent to the application considered an entire unit of activity.
- **Session Mobility** comprises functions and mechanisms to support session mobility.

• **Mobility-aware Application Support** comprising functions and services offered to the mobility-aware applications.

• **Mobility-based Application Support** comprising functions and services offered to the mobility-based applications.

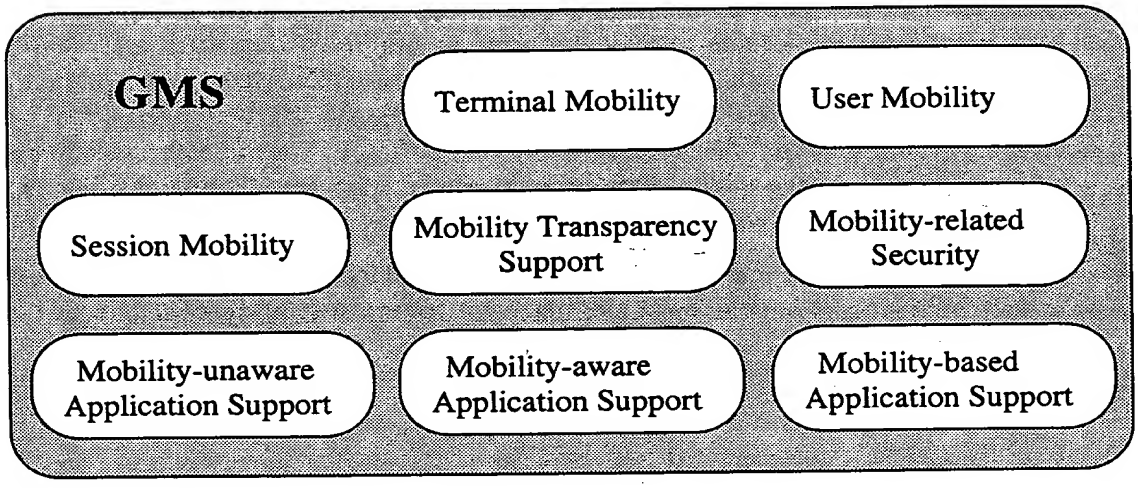


Figure 4.18 The Generic Mobility System and its functions

It is important to emphasize that the given overview of the GMS shows only a classification of the GMS functions. It is not a decomposition of the GMS into groups of objects. In fact, it is difficult and also not relevant to group the GMS objects according to the functions because several objects may cooperate to accomplish one function and one object may also support several functions. This will become clear as we proceed by identifying which objects are needed.

We shall in the following chapters successively study in details every functions of the GMS. For each function the required computational objects, their attributes and interfaces together with the interactions will be identified and considered thoroughly. In this way, the GMS will be gradually populated with computational objects. The whole process will only be concluded when all the GMS functions have been examined.

4.8 CONCLUSION

By offering mobility as a transparency in an ODP system, the design of mobile applications is no more complex than the design of fixed applications. Treating mobility as a transparency, the application designers do not need to be aware of the mechanisms necessary to deal with the different aspects of mobility.

Our proposal of separating all the functionality necessary to provide mobility both from both the network resource layer and from the application layer and grouping them in a separate functional layer called the Generic Mobility System, has several advantages. It promotes reusability in the sense that many components, mechanisms and algorithms can be used for different types of mobility. The separation of the functionality related to mobility from the application services increases flexibility: the mobility functionality can be modified without affecting the applications or the supporting network. Similarly, mobility support will be independent of the technological evolution of the network infrastructure. Another flexibility is due to the generic characteristic of the GMS: the GMS can be composed, customized, instantiated to suit the requirements of different telecommunications systems. By using the GMS as a unique and global mobility processing functionality, some unwanted service interactions occurring in present implementations of personal mobility such as a UPT service on IN, can be avoided [Do96b].

However, in order to design and implement an efficient GMS several issues must be studied and solved. This will be considered step by step in the following chapters.

5

Supporting Terminal Mobility

5.1 THE SCENARIO

We have shown that for terminals containing a DPE terminal mobility depends upon the interoperability between a mobile DPE node and a fixed DPE node. Terminals not containing a DPE will not be considered in this thesis. Let us study now what is needed to support access and location transparencies across several DPE platforms where continuity of interaction is not the case in general.

Consider two computational objects CO_t and CO_n, residing respectively on a mobile terminal and a processing node of the telecom system domain. We will use the same domain concept as before. To establish a stream between CO_t and CO_n, a request can be issued to the CSM (Communication Session Manager) specifying only the references of CO_t and CO_n (i.e their names) and the QoS parameters (bandwidth, error rate, etc.). This end-to-end connection specification is referred to as the Logical connection graph in TINA Connection Management Architecture [TIN95b].

From the reference of the computational objects, the CSM will deduce the corresponding TCSMs (Terminal Communication Session Manager) [TIN95b] and interact with them to get the identities of the corresponding NTPs (Network Termination Point) of the transport network used for stream flows. The TCSMs proceed to establish intra-node connections which is referred as Nodal Connection Graph in TINA Connection Management Architecture [TIN95b]. As shown in Figure 5.1, $TCSM_t$ connects CO_t and TTP_X (Terminal Termination Point). $TCSM_n$ connects NTP_Z and CO_n. Meanwhile, the CSM orders the CC (Connection Coordinator) to connect NTP_Y and NTP_Z. This is referred as the Physical Connection Graph. A stream is then established between CO_t and CO_n.

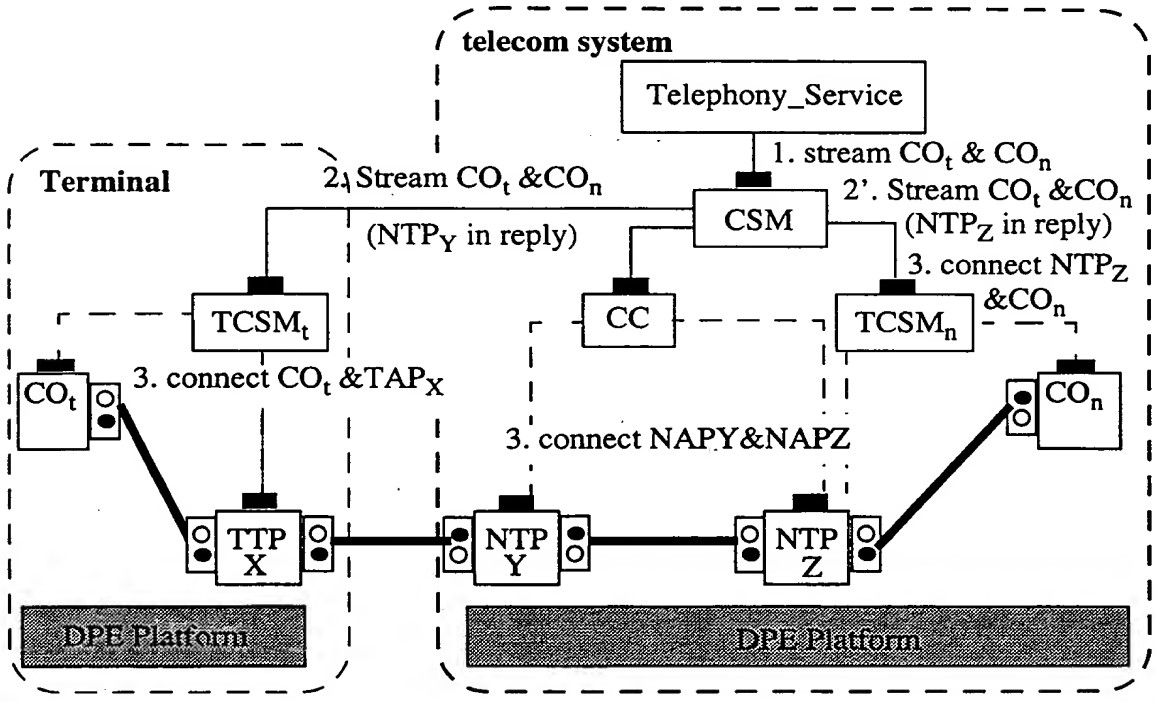


Figure 5.1 Stream establishment across two domains.

The condition for the success of the stream binding between one object residing in the mobile terminal and another one in the telecom system domain is the availability of operations between the CSM and the TCSM, i.e the CSM can invoke a `Stream_connect` operation on the TCSM. The CSM can always reach the $TCSM_n$ because it is always at the same place. If the TCSM is moving, as in the case of $TCSM_t$, it is not certain that CSM can always reach it.

5.2 ENABLING OPERATIONS BETWEEN THE MOBILE TERMINAL AND THE TELECOM SYSTEM

Let us now study how the DPE can support operations between two remote objects. Prior to any interaction (operation invocations), a binding must exist between the two objects. This is called implicit binding, i.e objects do not explicitly request establishment of the binding but the establishment is done by the DPE on the kernel Transport Network (kTN). This kTN is logically separated with the Transport Network supporting stream flows. Such a separation allows evolution of kernel communication in-

dependently of the evolution of the technologies used for stream flows transport [Aud96].

According to the TINA Engineering Modelling Concept [TIN94a], end-to-end information transfers between DPE nodes is provided by the kernel Transport Network which interconnects the DPE platforms, see Figure 5.2:

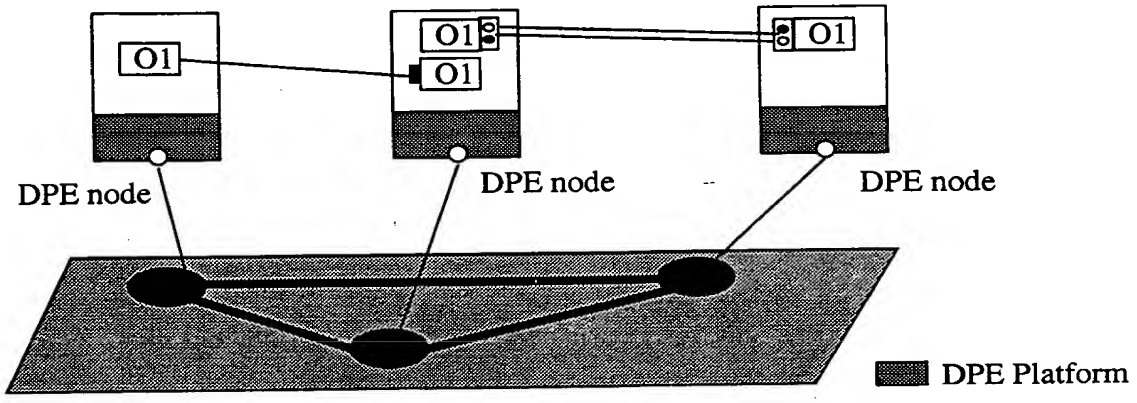


Figure 5.2 The kernel Transport Network

It is observed that operational interactions between computational objects are only possible if the kernel Transport Network is operative and there is always connectivity between two arbitrary DPE nodes, i.e every DPE node is reachable from every other DPE node.

For a wired network the connectivity is ensured at configuration, i.e necessary links between nodes are defined such that connectivity remains even if some nodes and links fails. The topology of the network is defined statically and will only be altered in case of a failure or reconfiguration. A DPE node can always determine how to reach another DPE node.

This is not always true with terminal mobility. The mobile terminal is a particular DPE node with special behaviour:

- It changes frequently the node with which it has direct link.
- It may just disappear from a node and reappear later at any other node.

The topology of the kernel Transport Network for systems containing mobile DPE nodes is changing dynamically and more seriously, it can also be in an undetermined state in the sense that the connectivity with such a mobile DPE node is not always ensured unless additional functionally is inserted in the DPE.

5.2.1 Our solution

We propose to consider the kTN as consisting of two parts:

- The fixed part comprising all fixed DPE nodes.
- The mobile part comprising all mobile DPE nodes.

At the boundary of the fixed part of the kTN there are several Network Access Points (NAP), i.e points where mobile DPE node (terminals) can connect themselves to the fixed kTN. It is worth noting that the NAP notion is different from the Network Termination Point (NTP) which designates the access point to the Transport network used for stream flows. As specified before the kTN and the Transport network are logically separate networks.

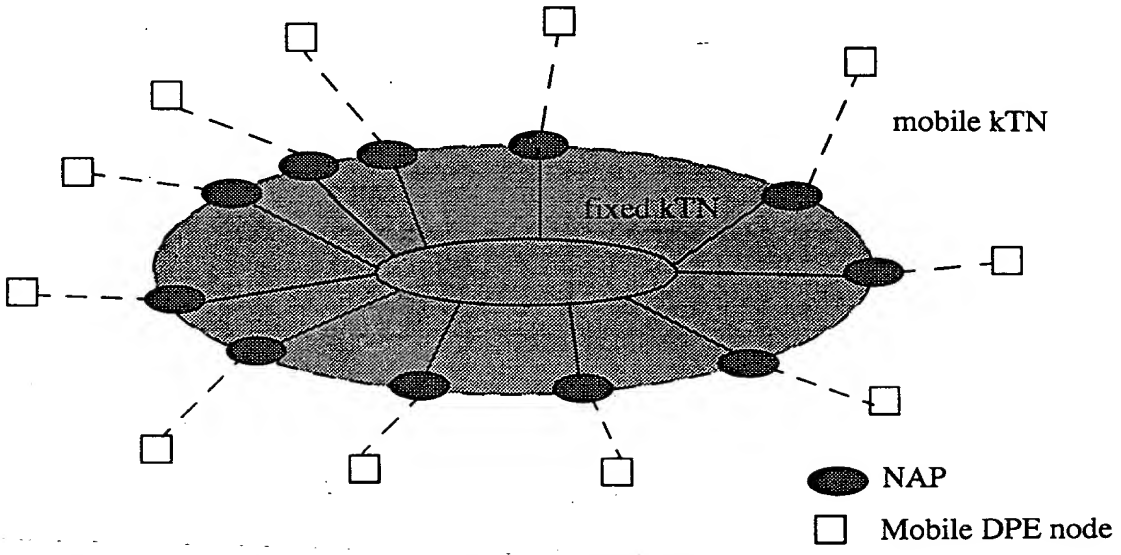


Figure 5.3 The kTN consisting of a fixed and a mobile part

An **NAP** object is introduced to represent a NAP on the kTN. A **NAP** object is an interceptor which stands at the boundary of the terminal domain and the telecom system domain and is responsible for checking, transforming and forwarding of interactions that cross the boundary. A **NAP** object has two communication interfaces, one with the mobile DPE and one with the fixed kTN. Several **NAP**s can be located at the same DPE node.

Each mobile DPE node may have one or several Terminal Access Points (TAP), i.e. the points where the mobile DPE node can exchange operations other DPE nodes. Here again, it is important to differentiate between a TAP and a Terminal Termination Point (TTP) which is used for the connection of stream flows. A TAP will be represented by a **TAP** object in the terminal domain

Prior to any operational interaction the mobile DPE, i.e one of its TAPs, must be attached to a NAP. Operational interactions between a computational object residing on a mobile DPE and a computational object residing on a fixed DPE always go through a TAP and a NAP.

Interactions can be divided into two types: interactions initiated by the fixed part and interactions initiated by the mobile part. Let us now examine both cases in more detail. Let two computational objects `CO1` and `CO2` reside respectively on a mobile DPE and a fixed DPE. `CO1` should be able to invoke any operation belonging to `I2` interface of `CO2` and vice versa, `CO2` should be able to invoke any operation belonging to `I1` interface of `CO1`, as shown in Figure 5.4.

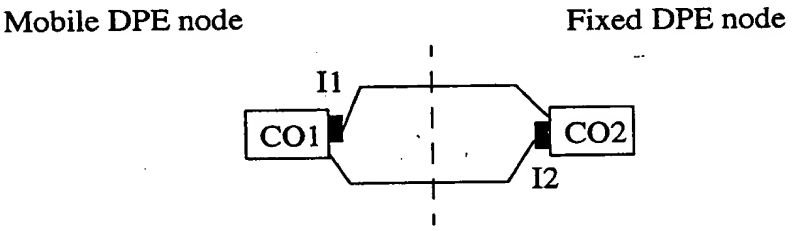


Figure 5.4 Interactions between COs residing on mobile and fixed DPEs

5.2.2 Interactions initiated by the fixed part

When `CO2` invokes an operation `opY` on `CO1`, the operation must be sent via the correct `NAP` object, i.e the one that is currently connected with the `TAP` of the mobile DPE. Since the terminal is moving, the `NAP` to which the `TAP` is connected may change from time to time.

To unburden `CO2` with mobility-related functions, a `Terminal_Agent` Object (`TA`) is introduced to undertake a kind of relocation function as shown in Figure 5.5. It keeps track of which is the current `NAP`. This relocation function is somewhat different from a relocation function defined in ODP. The latter records the change in location of an object when its is moved from one computing node to another. But in fact, recording the current `NAP` is semantically the same as recording the location of the terminal and the object `CO1` because the location of a terminal can be deduced from the `NAP`.

For each mobile DPE node (or terminal) one instance of the `Terminal_Agent` will be instantiated.

Instead of issuing an operation request directly to `CO1`, `CO2` issues a request to the `Terminal_Agent`. The `Terminal_Agent` will then forward it to the appropriate `NAP`. The `NAP` transfers it to the `TAP` which finally delivers it to `CO1`. To support in-

teractions initiated by the fixed part, the Terminal_Agent object is therefore required.

The introduction of the interceptor NAP and the Terminal_Agent in the fixed part and the TAP in the mobile part is sufficient to support operational interactions initiated by an object residing in the fixed part of the system.

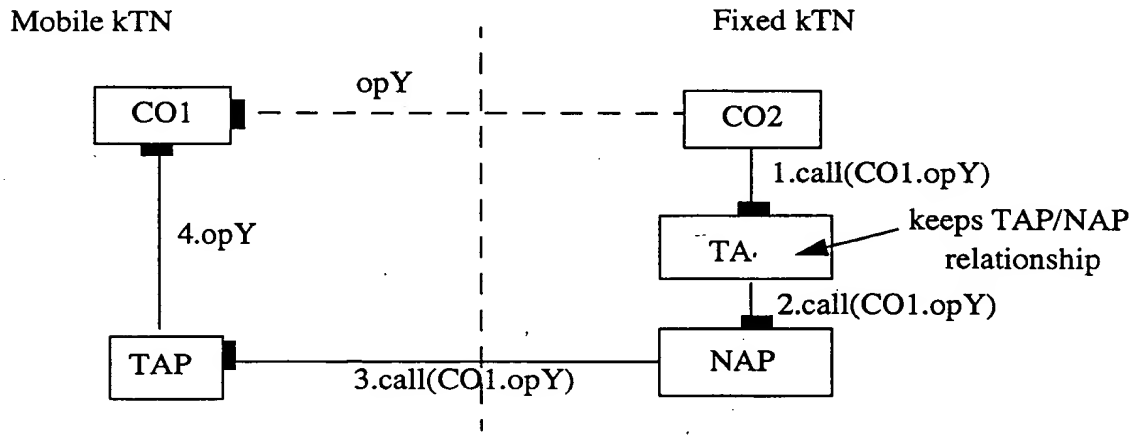


Figure 5.5 Interactions initiated by the fixed part

5.2.3 Interactions initiated by the mobile part

Suppose now that CO1 wants to invoke an operation opX on CO2 (Figure 5.6). First, the invocation must be conveyed to a TAP. This can be easily done assuming location transparency locally on the mobile DPE node. The TAP must then know to which NAP the operation should be forwarded. From the NAP the invocation can be conveyed to the TA (Terminal_Agent) and then to CO2, using access and location transparencies in the fixed kTN.

It is crucial that the TAP knows which NAP instance to communicate with in order to send the invocation to it. Since the TAP resides in a mobile DPE, the current NAP may be changing from time to time. The terminal mobility management consists precisely and partly of keeping track of the correct NAP instance. This responsibility is ensured by the SPA object (Service Provider Agent) introduced in Paragraph 4.5.5.

The SPA is thus entrusted with two responsibilities: supporting security functions and keeping location information. In this way, only one interceptor object is required in the terminal for managing both security and location updating. The introduction of the SPA is also convenient to keep the TAP hidden from the application objects. Instead of issuing an operation invocation to CO2, CO1 issues a request to the SPA. The reason is that a mobile DPE for example on a PABX may in fact have several TAPs which should be transparent to the application objects. The SPA will ensure

this transparency. Interactions initiated from the mobile DPE are realised as shown in Figure 5.6:

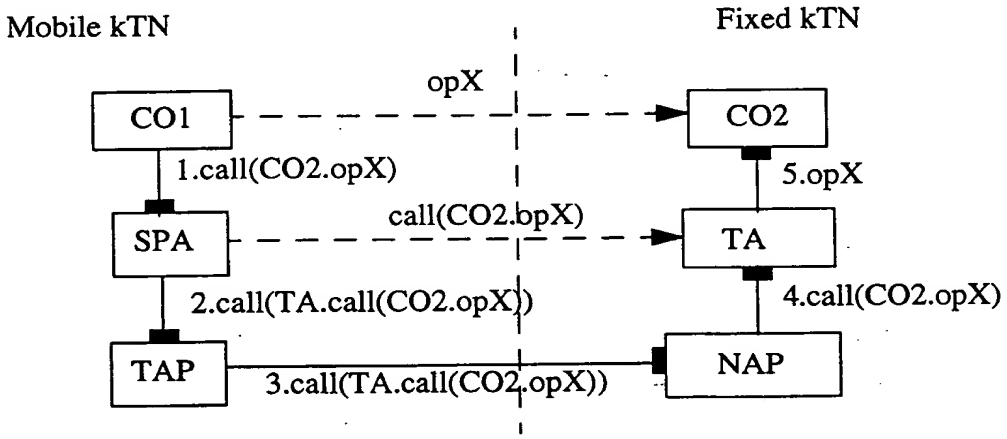


Figure 5.6 Interactions initiated by the mobile part

We may similarly use the **SPA** for interactions initiated by the fixed part in order to obtain a symmetric interceptor configuration. This gives us Figure 5.7 which replaces Figure 5.5 for operations originating in the fixed kTN.

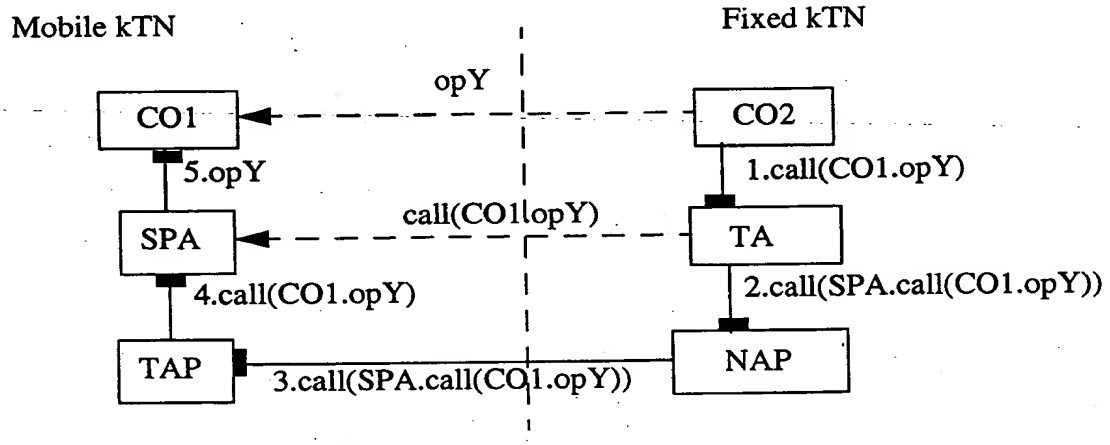


Figure 5.7 Interactions initiated by the fixed part - final version

It is observed that the condition for successful interactions initiated by objects on the mobile side is the definition of the association between the **TAP** and the **NAP**. Upon request from the **SPA**, the **TAP** will try to get in touch with a **NAP**. If it fails, i.e the

association is not defined, no interaction is possible and the terminal is out of contact with the fixed part of the network.

For interactions initiated by objects on the fixed side the success relies on two conditions. First, both the association between the TA and the NAP and the association between the NAP and TAP must be defined. If one or both of the two associations are undefined, the interaction will be unsuccessful. Second, the two associations must be consistent with each other. If the TAP is associated with an instance of NAP, then the TA must be associated with the same instance of NAP. An inconsistency will lead to failure.

Before continuing, some clarification of terminology is necessary to avoid confusion.

- The association between the TA and NAP is said to be *defined* when the TA is associated with an instance of NAP.
- The association is said to be *undefined* when the TA is not associated with any NAP.
- The *definition of the association* between the TA and the NAP means to associate an instance of NAP with the TA.
- The *removal of the association* between the TA and the NAP means that the TA is no longer associated with a NAP instance.
- The *redefinition of the association* between the TAP and the NAP means to associate the TA with another instance of NAP.
- The *determination of the association* between the TA and the NAP means to find out whether the association is defined or undefined.

Similar definitions apply for the association between the TAP and the NAP.

When the mobile DPE is moving, the TAP - NAP association and the NAP - TA association must change correlatively and may sometimes be undefined. The operations necessary to determine these two associations are commonly referred as location registration and location deregistration [GSM94b]. The various methods used to determine these two associations constitute therefore the different strategies for location registration and location deregistration. Below we shall look at several such methods and identify how they can be supported by the objects proposed above. Some of the methods will require additional objects. However, we will show that the basic structure proposed above will be returned also in cases.

5.2.4 Location registration and deregistration

5.2.4.1 The "on the fly" method

For interactions initiated by the mobile part, the association between the TAP and NAP can be determined "on the fly", i.e upon operation request the TAP may start searching for a NAP. If it finds a NAP a connection can be established towards this NAP. If a NAP cannot be found, it means that the mobile DPE has simply moved out of coverage area in the case of wireless terminal or is unplugged from the network in the case of wireline network.

There is no need to predetermine the association between TAP and NAP for interactions initiated by the mobile part: the "on the fly" method ensures that a NAP is found whenever there is one available.

For interactions initiated by the fixed part, the association between the TA and the NAP must be determined first and thereafter the association between the NAP and the TAP. This can also be done "on the fly". The TA will issue a broadcast to all the NAPs or requesting them in sequence, asking them to search for the mobile DPE. If none of the NAPs succeed, then the terminal is out of coverage or unplugged from the network; and nothing else can be done. If one NAP succeeds in finding the mobile DPE, then the TA can send operation invocations to this NAP which forwards them to the corresponding TAP. Interactions have then been enabled.

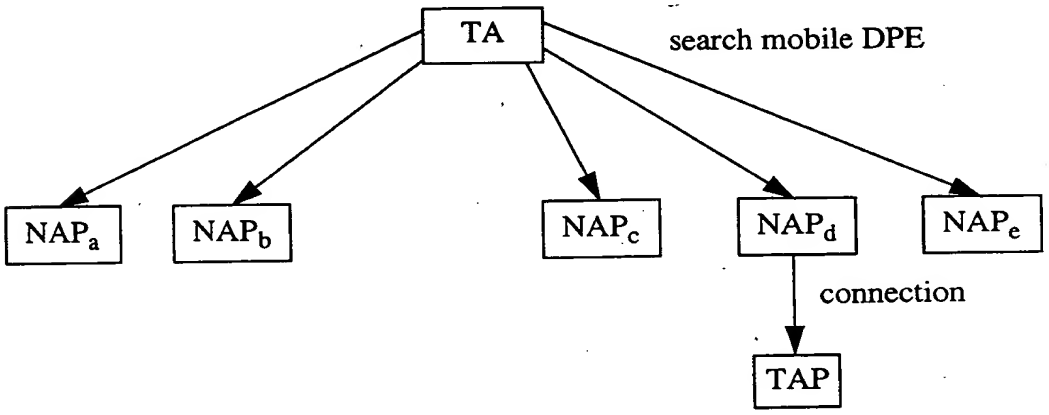


Figure 5.8 The "on-the-fly" searching method

This method is acceptable for small and "less geographically distributed" system, i.e system with small number of terminals and small number of NAPs. It can be time consuming for larger or more geographically distributed systems. It may take a while to find the terminal or to know that it is not possible to find it. This method requires also much activity in the telecom system domain. If there are **n** terminals and **m** NAPs in the system then in the worst case, **m x n** search procedures will be initiated simultaneously to find all the **n** terminals. Another inconvenience is due to mobility of the terminal. The terminal may move to another NAP right after the detection. Hence, when the operation invocation arrives at the NAP, the mobile DPE may have already moved to another NAP.

The method is, however, used on LANs and MANs to access terminals. This method is also used in GSM but then in combination with the method where a group of NAPs is first identified and the "on the fly" method is use to select the appropriate NAP within the group.

5.2.4.2 Methods based on the predetermination of the association between the TA and NAP

Other methods are based on the predetermination of the association between the TA and the NAP. The TA knows in advance whether the mobile DPE has contact with any NAP or not, and with which NAP instance it is associated. In other words, the TA has the ability to store the association between the TA and NAP. A possible implementation is to use a NAP pointer as shown in Figure 5.9.

Figure 5.9 The TA having a NAP pointer

The questions now are: *How can the* TA *get the information about the* NAP *and when should it update its* NAP *pointer*?

- The "TA's initiative" method

The immediate solution is that every TA asks periodically all the NAPs about the mobile DPE and updates its NAP pointer. If the searching procedure is done often enough, then the association between the TA and the NAP stored in the TA coincides with the real association. A terminal may therefore be in one of two states: *registered* when the association between the TA and the NAP is defined, and *deregistered* when it is undefined.

This method is similar to the "on the fly" methods but yields shorter response time. The status of all the terminals are always known by the telecom system domain, i.e whether they are registered and where they are. This method may be a good choice for systems where the status of the terminals play an important role such as fleet management, taxi dispatch, ambulance service, etc. Generally, this solution fits for small and "less geographically distributed" systems but may generate tremendously much activity in larger systems.

It is observed that the association between the TA and the NAP depends on the association between the TAP and the NAP. Only when the second association changes does the first one change and it should change as quickly as possible in order to avoid inconsistency between the two associations. Hence, the change of the association between the TAP and the NAP can be used to trigger the updating of the association between the TA and the NAP, i.e updating the NAP pointer of the TA.

The problem now is how to do the surveillance of the association between the TAP and NAP in an efficient way. The association between a TAP and a NAP is defined

when all the interactions from and to the mobile DPE having this TAP can go through the corresponding NAP. It is undefined when interaction from and to the mobile DPE cannot go through any NAP. At this stage, there is a difference between wireline systems and wireless systems. In wireline systems, the terminal is connected to the telecom system domain via copper lines or optical fibers on a permanent basis. In wireless systems the connection is done by radio frequency or infrared link. The methods to determine the association between the TAP and the NAP are different in the two cases and need to be considered separately.

5.2.4.2.1 The wireline case

In the wireline case, the association between the TAP and the NAP is directly reflected by and is equivalent to the physical link between the mobile DPE and the telecom system domain. The association between the TAP and the NAP is also equivalent to the association between the TA and NAP. We have:

Physical link <=> Association TAP - NAP <=> Association TA - NAP

A change in state of the physical link results in the change of the association between the TAP and the NAP and consequently a change of the association between the TA and the NAP. The wireline terminal can be in one of two states: *registered* when the physical link is up and the association between the TAP and NAP and the association between the TA and the NAP are defined; *deregistered* when the physical link is down and both associations are undefined. There is no intermediate state.

Terminal state	**Physical link**	**Ass.TAP - NAP ∧ Ass. TA - NAP**
registered	Up	Defined
deregistered	Down	Undefined

Figure 5.10 States of the wireline terminal

- The "physical link surveillance" method

A change of the physical link can be used to trigger a transition in terminal state and a location registration or deregistration. The transition of the terminal state is shown in figure below:

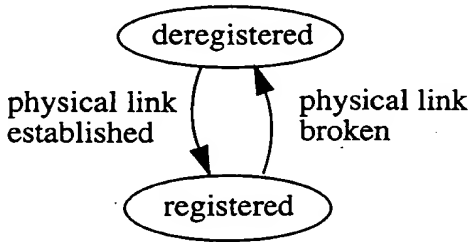


Figure 5.11 The state transition diagram of a wireline terminal

When the physical link is broken, i.e the terminal is unplugged from the socket or switched off or the line is broken, the association between the TAP and the NAP is removed. The NAP will immediately discover the situation and notify the TA. The TA set its NAP pointer to Nil. From now on, any attempts to invoke operations on the mobile part will be denied by the TA. The terminal is in the *deregistered* state.

When the physical link is established, i.e the terminal is plugged into the socket or switched on, the association between the TAP and the NAP is defined. The NAP will immediately discover the situation and notify the TA. The TA set its NAP pointer to the corresponding NAP. Interactions are then enabled. The terminal is in the *registered* state.

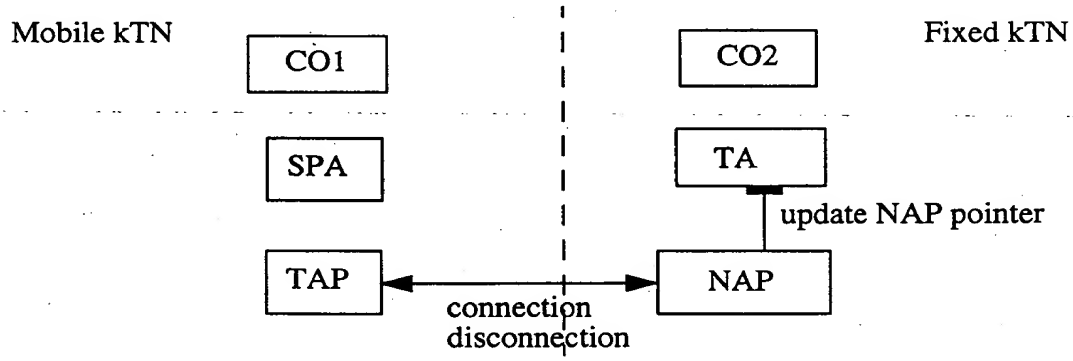


Figure 5.12 Location updating for wireline terminal

5.2.4.2.2 The wireless case

The wireless terminal stays most of the time in the disconnected state. When the physical link is down, it does not necessarily mean that the association between the TAP and the NAP is removed. The mobile DPE may still be there and, upon request,

the physical link can be reestablished. The mobile DPE may also have left the area. Hence, the state change of the physical link does no longer reflect the state change of the association between the TAP and the NAP.

Other methods to detect and update the change of the association between the TAP and the NAP are required. There are two alternatives: either the initiative is taken by the NAP or by the mobile DPE.

a. Initiative by the NAP

- The NAP's initiative method

The detection of the changes of the association between the TAP and the NAP means that the NAP is able to find out which mobile DPEs are located in its area. The NAP must therefore have the capability to store the identifier of the TAPs which have been previously connected to it, i.e before the physical link is "broken".

Periodically, the NAP broadcasts a "hello" in its area. Any mobile DPE present in the NAP's area will respond to the broadcast with its TAP identifier. The NAP will then compare the received TAP identifiers with the stored ones. For mobile DPEs previously registered, there is no change in the state of the association between the TAP and the NAP and no action is necessary. For mobile DPEs which have just arrived, the NAP issues an update call to the corresponding TAs. The TAs will set their NAP pointer to the NAP. For mobile DPE which has left, the NAP also initiates an update of the corresponding TAs indicating that the mobile DPE did not respond. The TAs will reset their NAP pointer to Nil.

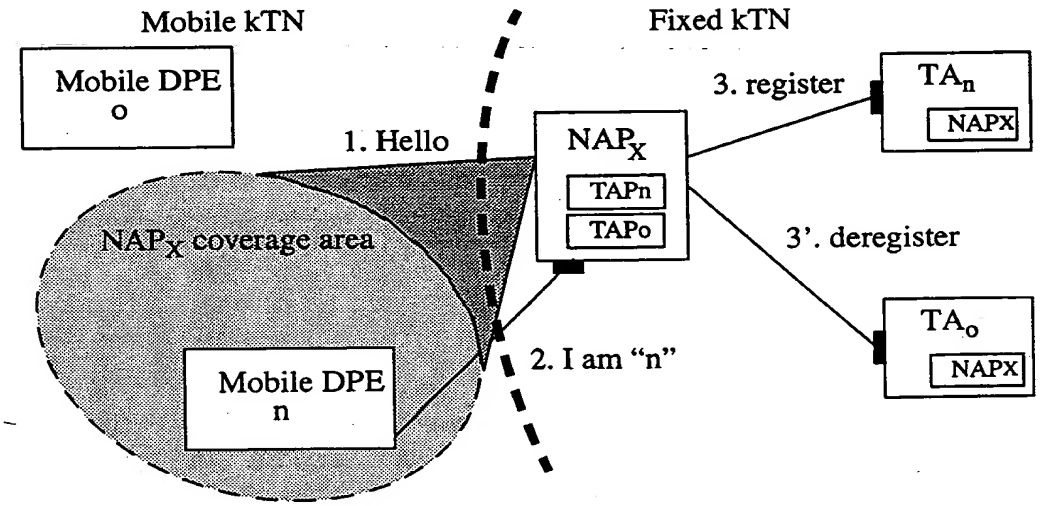


Figure 5.13 The case of detection function assumed by the NAP

If the intervals between consecutive detection processes are short enough, the representation of the association between the TAP and the NAP in the telecom system domain can be considered to be in agreement with the real association. The association is also determined for all the terminals, i.e the telecom system domain has the status of all the terminals. A terminal is registered when the TAP and the NAP association is defined and deregistered when the association is undefined. In registered state all interactions from and to the mobile DPE are possible. In deregistered state no interaction is possible.

This solution can be used for systems where the status of the terminals are important. It can be used in systems with larger number of terminals but less geographically distributed, i.e small number of NAPs. For systems with large number of NAPs, this solution has many disadvantages. First, the NAPs must have storage capacity and consequently must keep the stored data, i.e TAP identifiers of mobile DPEs present in its area, up-to-date and consistent. Second, there are periodically much processing activity in every NAP. The third disadvantage is the inefficient use of the radio frequency or infrared access channel to the NAP which is a scarce resource shared by all mobile DPEs for both data transmission and signalling. When all the present mobile DPEs answer, much capacity is used just for detection of the DPEs.

b. Initiative by the mobile DPE

- The Periodic method

The detection and updating of the change of the association between the TAP and the NAP can also be initiated by the mobile DPE. Each mobile DPE can periodically report itself to the nearest NAP. The NAP will then send a register request to the corresponding TAs. In order to detect the silent terminals, i.e those that have disappeared without a deregistration, each TA can be equipped with a timeout. If a mobile DPE does not register itself within a period of time t_0, its TA will set it as deregistered. By this method, the status of all the terminals are always known by the telecom system domain. This method is suitable for systems with low number of terminals and more geographically distributed (larger number of NAPs).

It generates, however, activity both on the access channel and in the telecom system domain.

- The method based on location changes

It is observed that the changes of the association between the TAP and the NAP are caused by the mobility itself. If the mobile DPE knows that it has moved from one NAP to another, it also knows that the association between the TAP and the NAP has to be updated. The mobile DPE must have the capacity to store the identifier of the previous NAP and the capability to get the identifier of the current NAP.

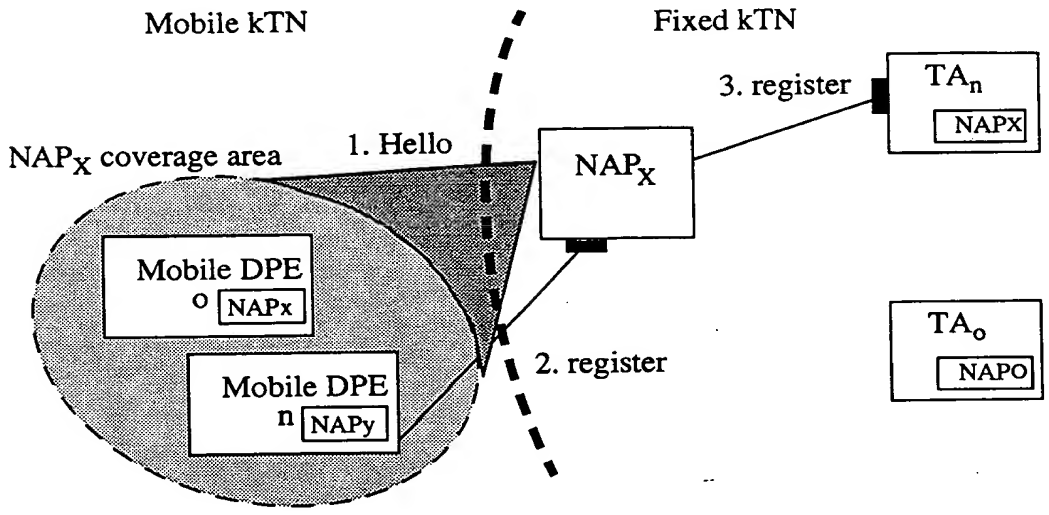


Figure 5.14 The case of detection function assumed by the mobile DPE

As shown in Figure 5.13, a **NAP** is associated with a geographic area called **NAP** coverage area. The **NAP** broadcasts constantly its identifier in this coverage area. Every mobile station in the coverage area reads periodically this **NAP** identifier and compares it with the one stored previously. If they are similar, there is no change in the association between the **TAP** and the **NAP** and no updating is necessary. If they are different, there is a change in the association. The mobile DPE updates its local **NAP** pointer and requests a `register` operation of its **TA**. The request is sent to the NAP which forwards it to the corresponding **TA**. The **TA** sets its **NAP** pointer to the current **NAP**.

This alternative requires that the mobile DPE has some storage capacity and some intelligence to decide whether a registration is necessary or not. On the other hand, the **NAP** does not have to store and process the information about the mobile DPEs present in its area. The use of the access channel to detect and update the change of the association between the **TAP** and the **NAP** is more optimal since the probability that all the mobile DPEs will change **NAP** is very small and hence a total registration of all DPEs is very seldom.

The solution used in GSM [GSM94c] is slightly different from the one presented above. A base station can be mapped to a **NAP** in our model. There is however an intermediary object between the **NAP** and the **TA**. As shown in Figure 5.15 this object can be modelled as a group of **NAP**s, **GNAP** or as a "mirror" **TA**, **MTA** holding some data of the **TA**. The location updating is executed only when the mobile station has moved from one **GNAP** or **MTA** to another. The determination of the **NAP** is done using the "on the fly" method.

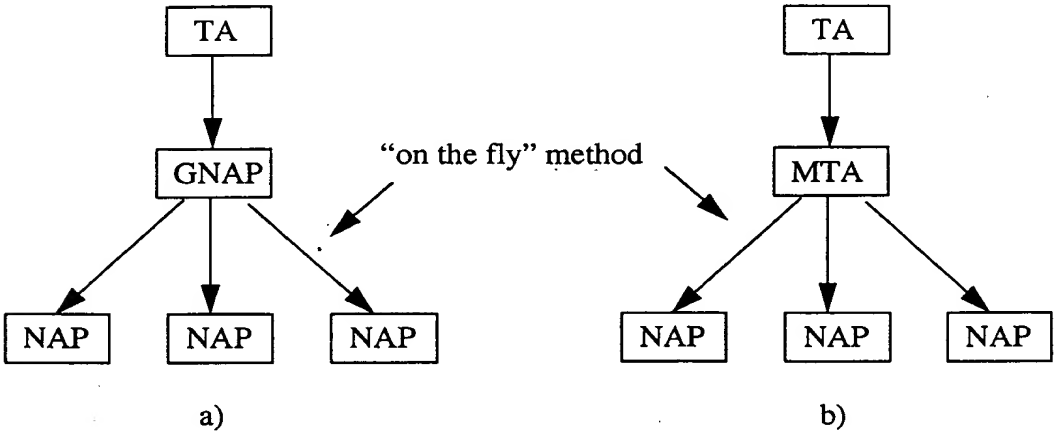


Figure 5.15 Alternative models of GSM

There is, however, a problem. The updating of the association between the **TAP** and the **NAP** and the association between the **TA** and the **NAP** are initiated by the mobile DPE when it knows that a change has occurred. It is not always true that the mobile DPE is aware of such a change and capable of notifying the telecom system domain. It can be switched off and brought to another area, a fault may occur in the mobile DPE, or it may have run out of battery power. It can also move out of the coverage area, loose all contact with the telecom system domain and is unable to initiate the updating.

In such a situation there may or may not be a mismatch between the real association between the **TAP** and the **NAP** and the perception of the telecom system domain, i.e the state of the association stored in the telecom system domain. When a terminal is registered, i.e the associations between the **TA** and the **NAP** and the association between the **TAP** and the **NAP** are defined, it is not guaranteed that interactions from and to the mobile DPE are possible. Only when the physical link is in operation, i.e there is activities to or from the terminal, is the state well-defined. When the physical link is down, nothing is known about the location of the mobile DPE.

In the telecom system domain, it is useful to differentiate the two registered states of the terminal, namely **registered and confirmed** state and **registered and unconfirmed** state. In addition to the **NAP** pointer the **TA** needs now also to save the terminal state. On the mobile DPE side there is no need for additional state but the registered state and the deregistered state. The **NAP** pointer is sufficient to represent these two states.

Seen from the telecom system domain, a wireless terminal may therefore be in one of the following three states:

- **deregistered**: the mobile station has actively notified that it has been taken temporarily out of service. It can also be faulty, out of battery or out of sys-

tem coverage.

- **registered and unconfirmed**: the mobile station is registered at a NAP but there is no activity on it, i.e the physical link is down.
- **registered and confirmed**: the mobile station is registered at a NAP and there is activity going on, i.e the physical link is up.

The states of the wireless terminal is thus defined by the state of the physical link, the association between the TAP and NAP and the association between the TA and NAP as follows:

Terminal state	**Ass.TAP - NAP ∧ Ass. TA - NAP**	**Physical link**
deregistered	Undefined	Down
registered & unconfirmed	Defined	Down
registered & confirmed	Defined	Up

Figure 5.16 States of the wireless terminal

From the registered and confirmed state the mobile DPE changes its state to the registered and unconfirmed state when all the activities have ceased. It will remain in the registered and unconfirmed state until some successful activity takes place between the mobile DPE and the fixed DPE. The terminal state is then reset to the registered and confirmed state. If the request is unsuccessful, the terminal state will be changed to the deregistered state and will remain there until a registration takes place. From the registered and confirmed state, the terminal state can be changed to the deregistered state when the mobile DPE has explicitly made a deregistration request or disappears.

The transitions between the terminal states are shown in figure below:

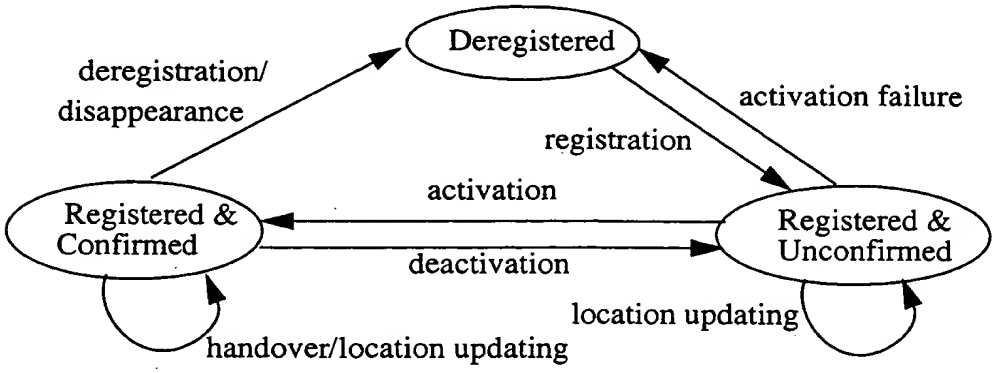


Figure 5.17 The state transition diagram of the wireless terminal

It is worth noting that in addition to normal transitions such as registration, activation, etc. there are two special transitions: handover and location updating. In fact, the handover and location updating can be regarded as a transition because the associations **TAP - NAP** and **TA - NAP** do change but remain defined. Location updating may be regarded as a transition from the registered and unconfirmed state back to the registered and unconfirmed state. Similarly, handover may be regarded as a transition from the registered and confirmed state to the registered and confirmed state. A handover may follow a location updating or may be executed alone. From one registered and confirmed state, the terminal may also migrate to another registered and confirmed state after a location updating alone without a handover. We shall now study all the transitions in details, except the transitions from registered and confirmed state to registered and confirmed state, which we will examine after the study of stream interactions since stream interfaces are involved in the transition.

a. Registration

Registration is the transition from the deregistered state to the registered and unconfirmed state. The transition happens when the wireless terminal is put back to service after having been powered off or out of coverage.

In the mobile DPE a new object called **Location_Mgr** is introduced. This object is responsible for registration when location changes occur. For this sytem to work, the **NAP** will broadcast its identifier to all the **TAP**s in its area.

Upon power up or activation or contact re-establishment, the **TAP** will get such a **NAP** identifier. It transfers it to the **Location_Mgr** through the operation `New_NAP()`. The **Location_Mgr** fetches the stored **NAP** identifier and compares it with the new one. Since in the deregistered state, the **NAP** pointer is set to `Nil`, a mismatch occurs. The **Location_Mgr** updates the NAPid and invokes the operation `Register()` on the **SPA**. The **SPA** invokes `TAP.Call(TA.Register())`. The **TAP** tries to connect itself to a **NAP** (through a radio link or infrared link). If it succeeds, the terminal and the telecom system domain will proceed with the security procedures, i.e identification, authentication and access control (These procedures

will be described in Chapter 7). If the security procedures are successful, the TAP will call the operation Call(TA.Register()) on the NAP. The NAP calls the operation Register() of the TA. The TA sets the Terminal_State to registered and unconfirmed and its NAP pointer to the corresponding NAP. The Location_Mgr sets also its NAP pointer to the corresponding NAP. The physical link between the TAP and the NAP can then be disconnected. In the case that one of the security procedures fails, the terminal will remain in the deregistered state.

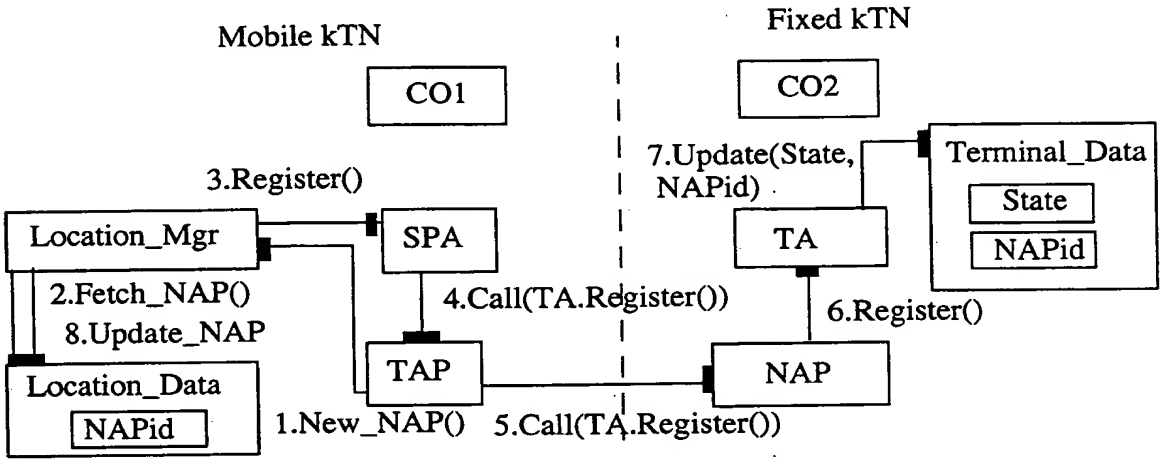


Figure 5.18 Transition from the deregistered state to the registered and unconfirmed state

In the computational model shown in Figure 5.18, two additional objects are introduced to represent the terminal location data on both sides. In the mobile DPE, the separation of the location data into the object Location_Data is not strictly necessary but is included for clarity. The location data may incorporated in the Location_Mgr. On the other hand, in the fixed DPE the separation of the location data into the object Terminal_Data is justified by the fact that these data must be persistent and hence saved in a different place than the TA itself. Indeed, in the Engineering viewpoint where distribution is taken into account, the distribution, replication and migration of the TA (processing code) and the Terminal_Data (data) will play a major role in the efficiency of the different location tracking algorithms of the terminal.

The procedure for the terminal registration is shown in Figure 5.19. In the figure, an announcement operation is represented by a single arrow while an interrogation operation is represented by a pair of arrows. The first arrow indicates the initiation of the operation and passing of arguments. The second arrow is the response returned by the server object to the client object. The response is denoted RXX() where XX is the name of the operation.

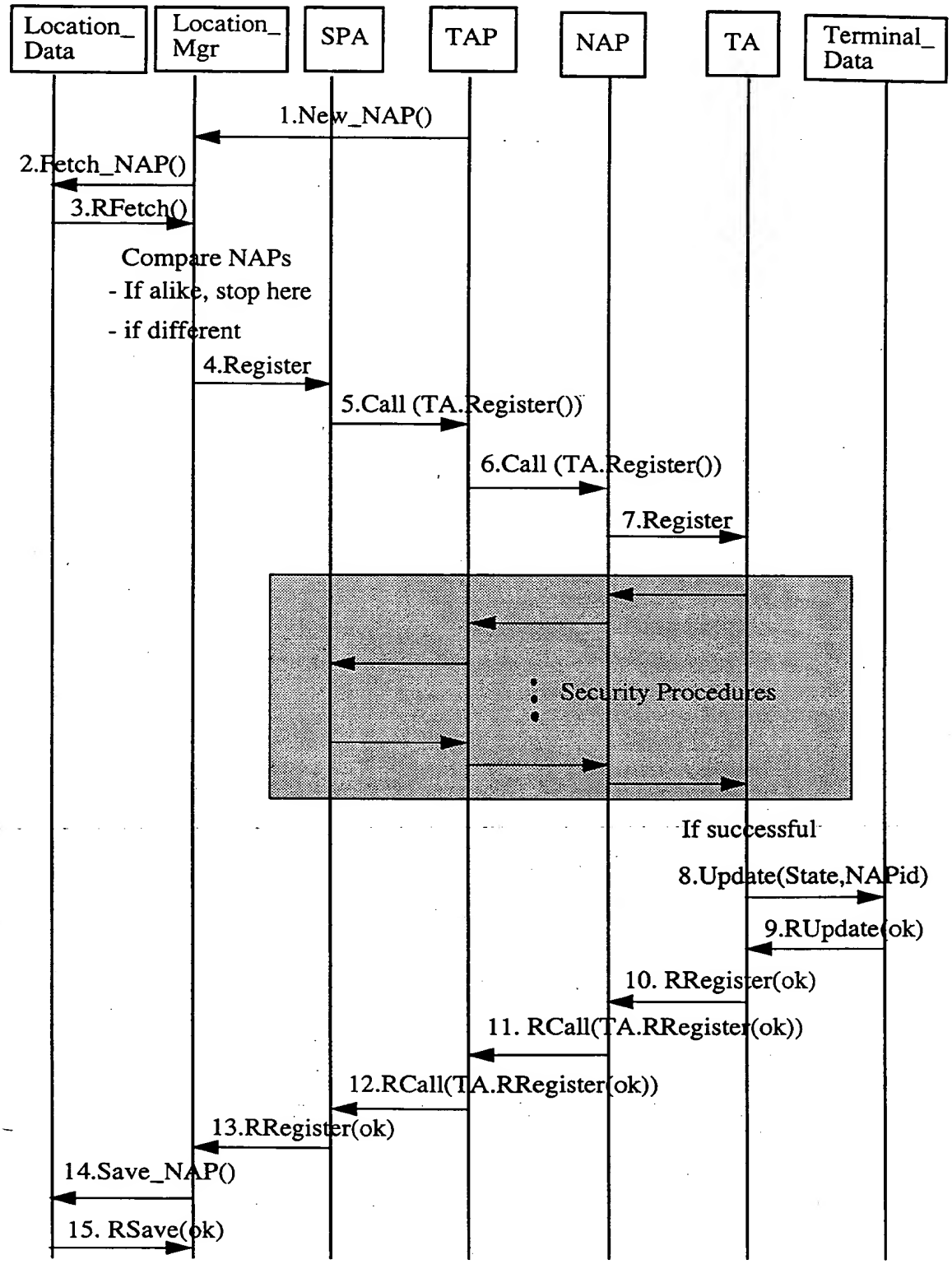


Figure 5.19 The procedure for the terminal registration

b. Activation

Activation is the transition from the registered and unconfirmed state to the registered and confirmed state. The transition occurs when an activity is initiated either from the mobile side or the fixed side. By activity request is meant both an operation request or a request for stream establishment. For operation request the **TA** just needs to update the **Terminal_State** to registered and confirmed. For stream request additional actions must be performed to enable the stream release later on.

We shall first study the two cases of activation by operation requests: the case of operation request initiated by the mobile side and the case of initiation by the fixed side. Activation associated with stream request will be considered later on.

Activation by operation from the mobile side

When an object **CO1** on the mobile DPE wants to invoke an operation OpX of an object **CO2** on the fixed side, it issues the operation Call(CO2.OpX()) of the **SPA**. The **SPA** will then invoke the operation Call(TA.Call(CO2.OpX())) on the **TAP**. The **TAP** then tries to connect itself to a **NAP**. If it is successful, the terminal and the telecom system domain will proceed with the security procedures. If these procedures are successful, the **TAP** will invoke the operation Call(TA.Call(CO2.OpX()) on the corresponding **NAP**. The **NAP** invokes the operation Call(CO2.OpX()) on the **TA**. Before calling the operation OpX() on the object **CO2**, the **TA** sets the **Terminal_State** to registered and confirmed. The **TA** will also set the Operation_State to "ON". The Operation_State is used to mark that an operation channel has been established between the mobile DPE and the telecom system domain. It will be shown later that such attribute in the **Terminal_Data** is necessary in the deactivation of the terminal.

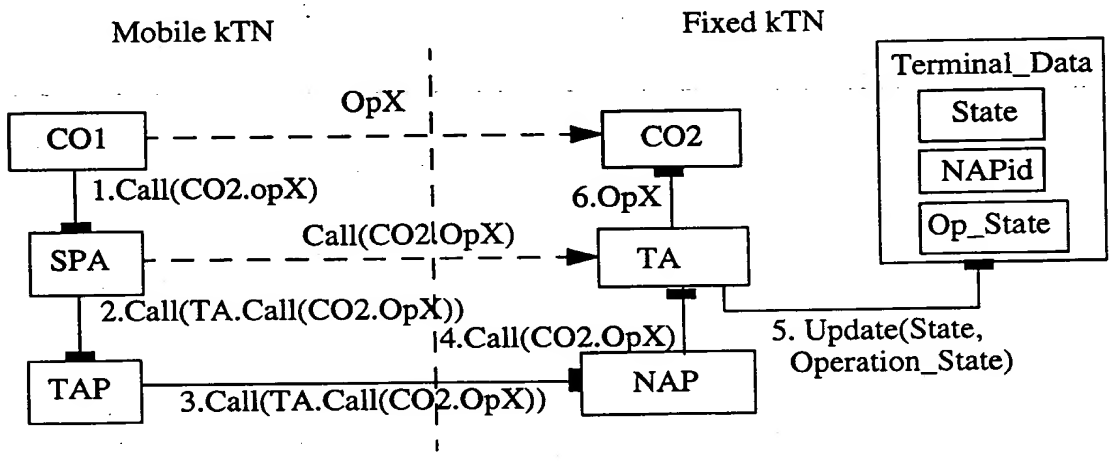


Figure 5.20 Activation by operation initiated by the mobile part

The procedure for activation initiated by the mobile part is shown in the following figure.

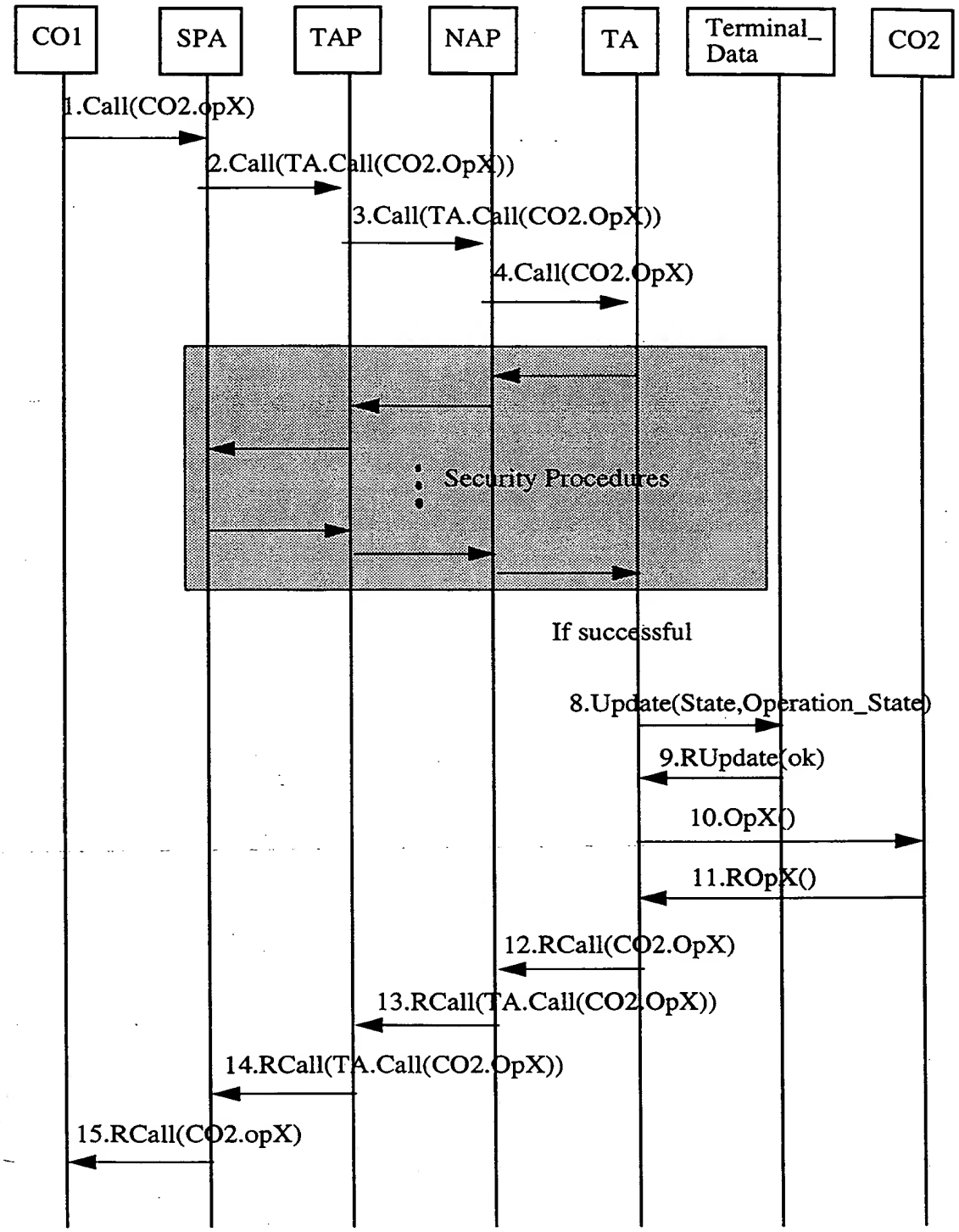


Figure 5.21 The procedure for the activation by operation initiated by the mobile part

Activation by operation from the fixed side

When an object `CO2` on the fixed DPE wants to invoke an operation `OpY` of an object `CO1` on the mobile DPE, it issues the operation `Call(CO1.OpY())` of the `TA`. The `TA` will initiate the security procedures. It these procedures are successful, the `TA` sets the `Terminal_State` to registered and confirmed. The `TA` will then call the operation `Call(TA.Call(CO2.OpY()))` on the `NAP`. The `NAP` will invoke the operation `Call(SPA.Call(CO2.OpY())` on the corresponding `TAP`. The `TAP` invokes the operation `Call(CO2.OpY())` on the `SPA`. The `SPA` then invokes `OpY` on `CO1`.

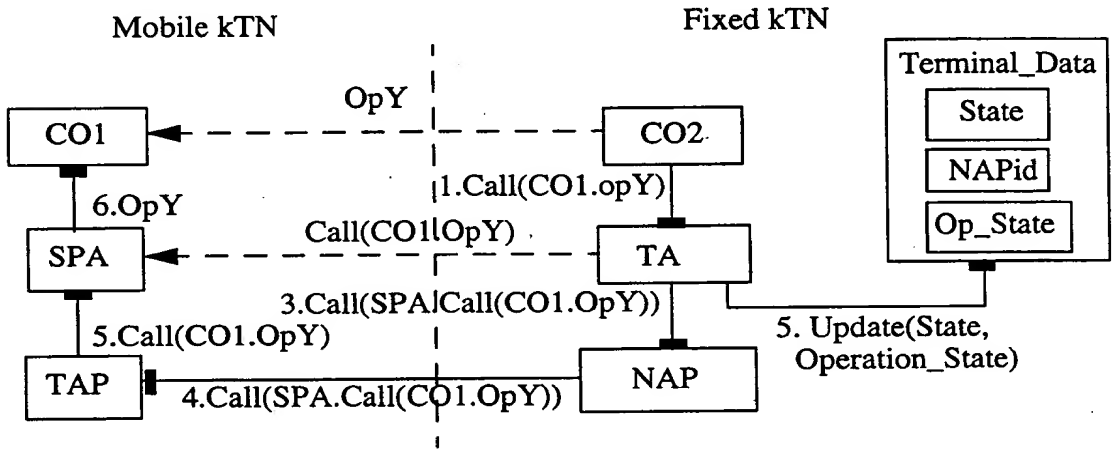


Figure 5.22 Activation initated by the fixed part

The procedure for activation initiated by the fixed part is shown in the following figure.

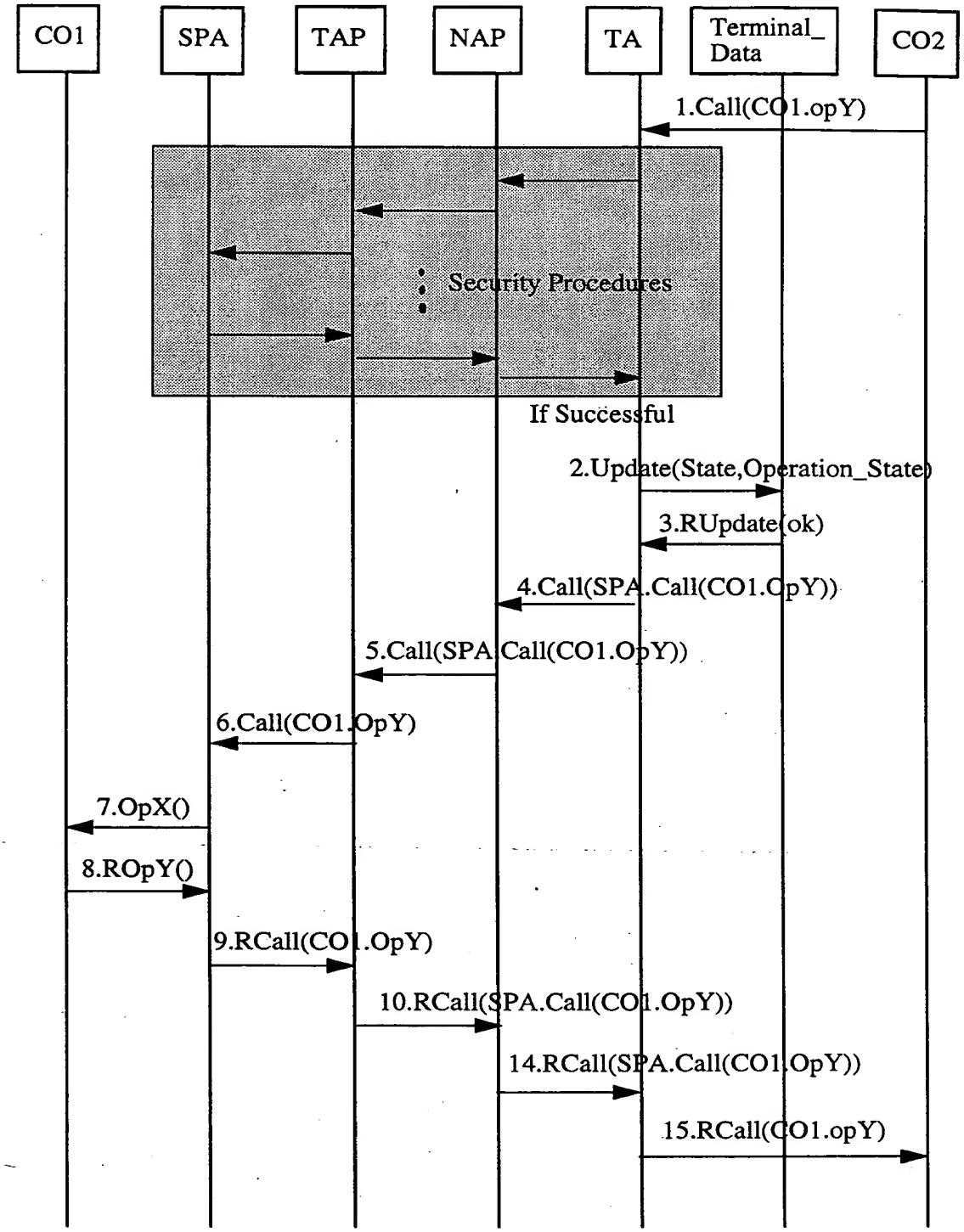


Figure 5.23 The procedure for the activation by operation initiated by the mobile part

Activation by stream request

As shown in Figure 5.1, the establishment and release of a stream between two objects are supported by the CSM (Communication Session Manager). In order to establish or release a stream between an object CO1 on the mobile DPE and an object CO2 in the fixed DPE, the CSM must interact with the $TCSM_M$ (Terminal Communication Session Manager) on the mobile DPE, the CC (Connection Coordinator) and the $TCSM_F$ on the fixed DPE node where the object on the fixed side is located. The Connect(CO2) operation on the $TCSM_F$ can be issued directly because $TCSM_M$ reside on the same DPE as CSM. On the other hand, as in the case of operation request to any objects on the mobile DPE, the request of the Connect(CO1) operation on the $TCSM_M$ must go through the TA, the NAP, the TAP and the SPA, as shown in Figure 5.24.

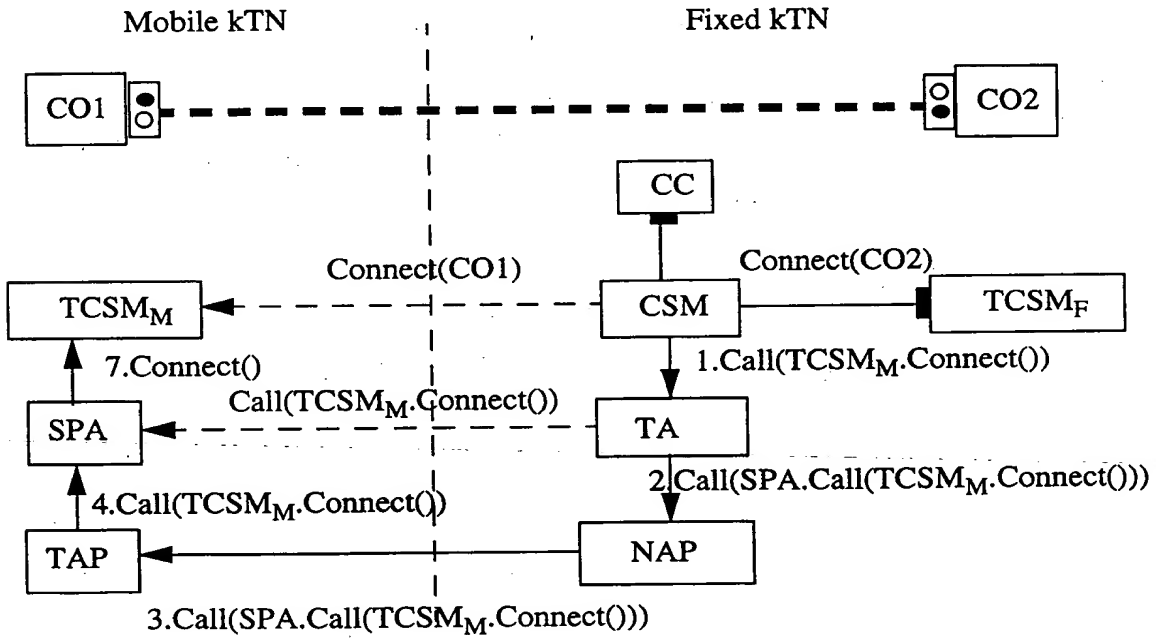


Figure 5.24 Operations involved in the stream establishment

The Connect() operation on the $TCSM_M$ is different from all other operations in the sense that activity between CO1 and CO2 is initiated by this operation. This activity will continue after the accomplishment of the operation. The activity continues, however, in the transport network and directly between the communicating objects CO1 and CO2 without going through the TA. The TA must somehow be informed about the stream activity.

Therefore, when the CSM invokes Call(TCSM$_M$.Connect()) on it , the TA must be aware that the call comes from the CSM and understand that a stream will be established. Similarly, when the CSM invokes Call(TCSM$_M$.Disconnect()), the TA must understand that a stream will be released.

The TA must treat the CSM differently from other objects invoking the Call operation. Before forwarding the invocation to the NAP by invoking Call(SPA.Call(TCSM$_M$.Connect()), the TA must interpret the arguments of the Call operation, namely TCSM$_M$ and Connect or Disconnect. Alternatively, a dedicated operational interface between the TA and CSM can be defined for the establishment and release of a stream. The second solution is less convenient since it requires that the CSM knows about the TA. We shall see later in Chapter 8 that the first solution is chosen because it enables the integration of the GMS into the system in a transparent way.

In addition, the Terminal_Data must have a table containing all the streams which have been established with objects on the mobile DPE. Such a StreamTable may contain field like:

ObjectNameX.StreamK

where ObjectNameX denotes the object on the mobile DPE and StreamK denotes the stream since one object may have several stream interfaces. An example of StreamTable is shown in Figure 5.25. The objects CO1 and E has one stream each while the object A has two streams.

ObjectNameCO1.Stream1
ObjectNameA.Stream1
ObjectNameA.Stream2
ObjectNameE.Stream1

Figure 5.25 Example of the table of streams

The implementation of the table can be realized in several different ways, e.g array or linked list and will not be studied in more details here. Upon receipt of Call(TCSM$_M$.Connect()), the TA will add an element to StreamTable. Upon receipt of Call(TCSM$_M$.Disconnect()) the TA will remove the corresponding element from the StreamTable.

Upon receipt of the operation Call(TCSM$_M$.Connect()), if the Terminal_State is already registered and confirmed, no updating is required; and

the security procedures need not be invoked. The **TA** just adds a new element to its StreamTable.

If the `Terminal_State` is registered and unconfirmed, the **TA** starts with the security procedures. If these procedures are accomplished successfully, the **TA** sets `Terminal_State` to registered and confirmed and adds a new element to the StreamTable. From here, the procedure is similar to the one defined for operations in general. The **TA** will then invoke the operation `Call(SPA.Call(`$TCSM_M$`.Connect()))` on the **NAP**. The **NAP** invokes the operation `Call(SPA.Call(`$TCSM_M$`.Connect()))` on the corresponding **TAP**. The **TAP** calls `Call(`$TCSM_M$`.Connect())` on the **SPA**. The **SPA** calls **Connect()** on the $\mathbf{TCSM_M}$.

The procedure for activation by stream request is as follows:

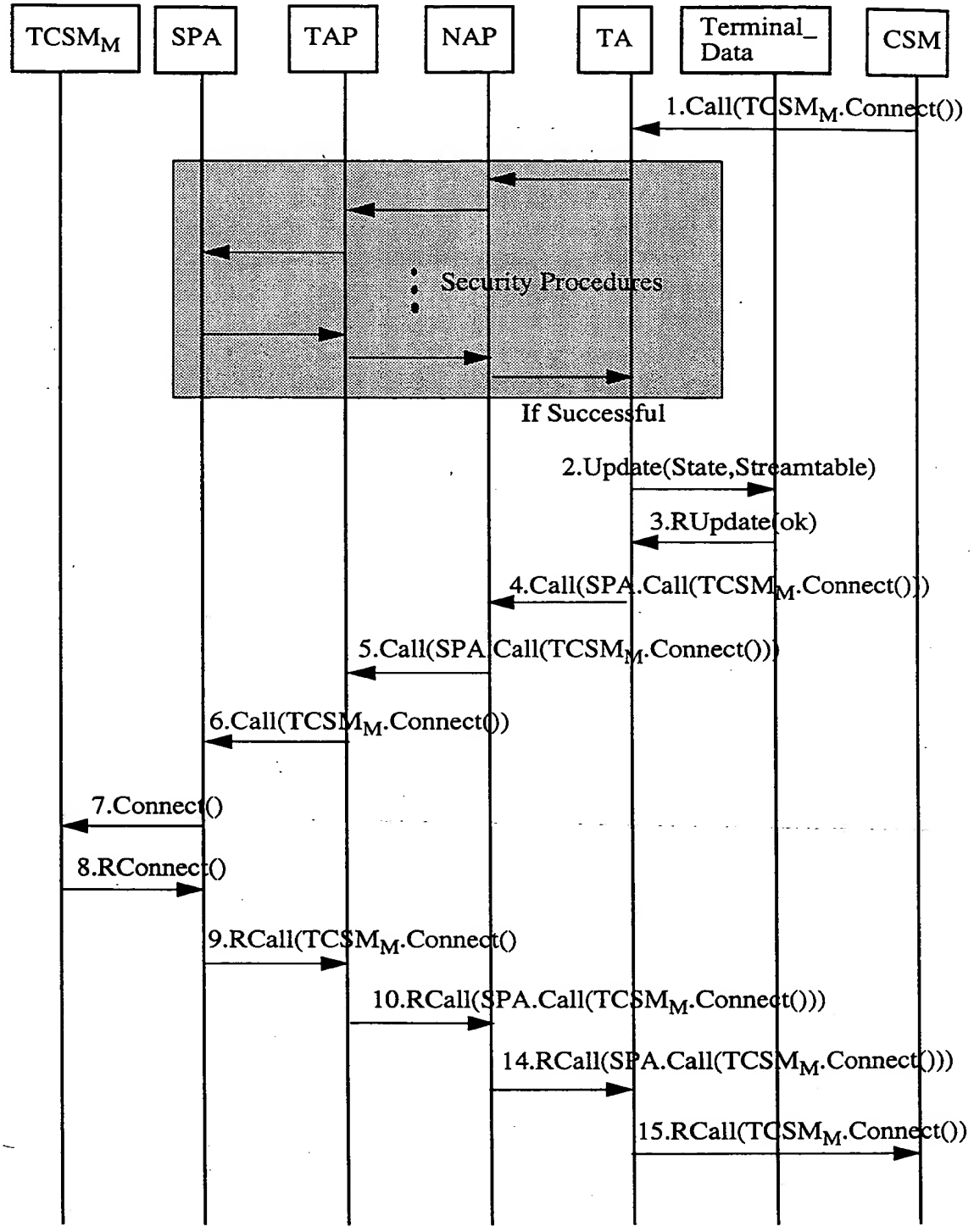


Figure 5.26 The procedure for the activation by stream request

c. Deactivation

Deactivation is the transition from the registered and confirmed state to the registered and unconfirmed state. The transition happens when all the activities have stopped, i.e there is no more operation or stream between the mobile DPE and the telecom system domain.

When there is no more operation between the mobile DPE and the fixed DPE, the TA will discover the situation since all the operations are conveyed through the TA. The TA will check the StreamTable. If it is not empty, i.e there is still ongoing stream flow, the TA will set the Operation_State to "OFF" and wait until the StreamTable becomes empty. If the StreamTable is empty, i.e all the streams have been released. The TA will invoke the operation OpChanRel() on the NAP. The NAP is the object responsible for the maintenance and surveillance of the operation channel with the mobile DPE. Upon receipt of the call OpChanRel(), the NAP will release the operation channel. The TA will then set the Terminal_State to registered and unconfirmed.l

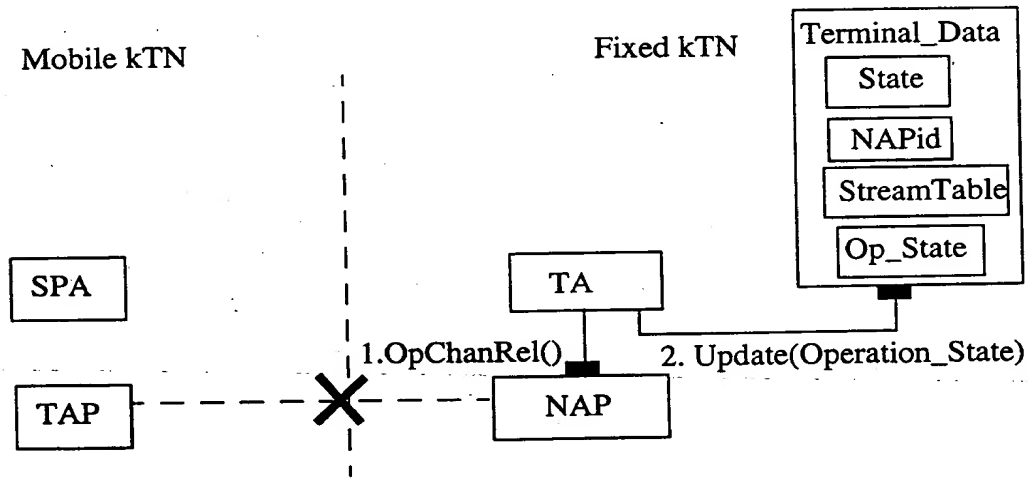


Figure 5.27 The release of the operation channel

For streams, the object responsible for the release is the CSM. To release a stream between an object on the mobile DPE and one on the fixed DPE, the CSM has to invoke the operation Disconnect() of the corresponding $TCSM_M$. As for other operations, the Disconnect() operation is conveyed through the TA. Upon receipt of Call($TCSM_M$.Disconnect()), the TA will remove the corresponding element from the StreamTable. If the StreamTable becomes empty and if the Operation_State is "OFF", the TA will set the Terminal_State to registered and unconfirmed.

d. Deregistration

Deregistration is one of the two transitions from the registered and confirmed state to the deregistered state. The transition occurs when the terminal requests explicitly a deregistration, e.g when the mobile DPE is powered off (GSM functionality [ETS94b]). This alerts the `Location_Mgr` which resets the `NAPid` to `Nil`. It will then call `Deregistered()` on the **SPA**. The **SPA** invokes `Call(TA.deregister())` on the **TAP**. The **TAP** forwards the operation to the **NAP** by `Call(TA.Deregister())`. The **TAP** calls `Deregister()` on the **TA**. The **TA** will check the `StreamTable`. If it is not empty, the **TA** will call the operation `Unbind()` on the **CSM** to release all streams which are established with objects on the mobile DPE. If the `StreamTable` is empty, the **TA** can proceed directly with the updating of the `Terminal_Data`. It sets the `Terminal_State` to deregistered, the `NAPid` to `Nil`, the `Operation_state` to "OFF". The **TA** can then invoke `RelChan()` on the **NAP**. The **NAP** releases the operation channel with the mobile DPE. From now on, there is no more contact with the terminal and communication requests will be rejected so long as the `Terminal_State` is set to deregistered.

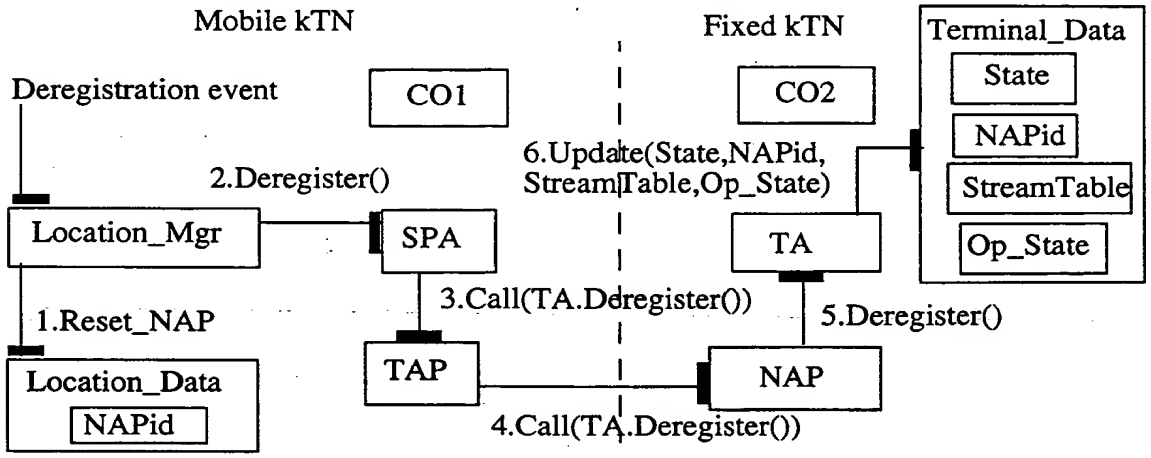


Figure 5.28 Deregistration - First transition from the registered and confirmed state to the deregistered state

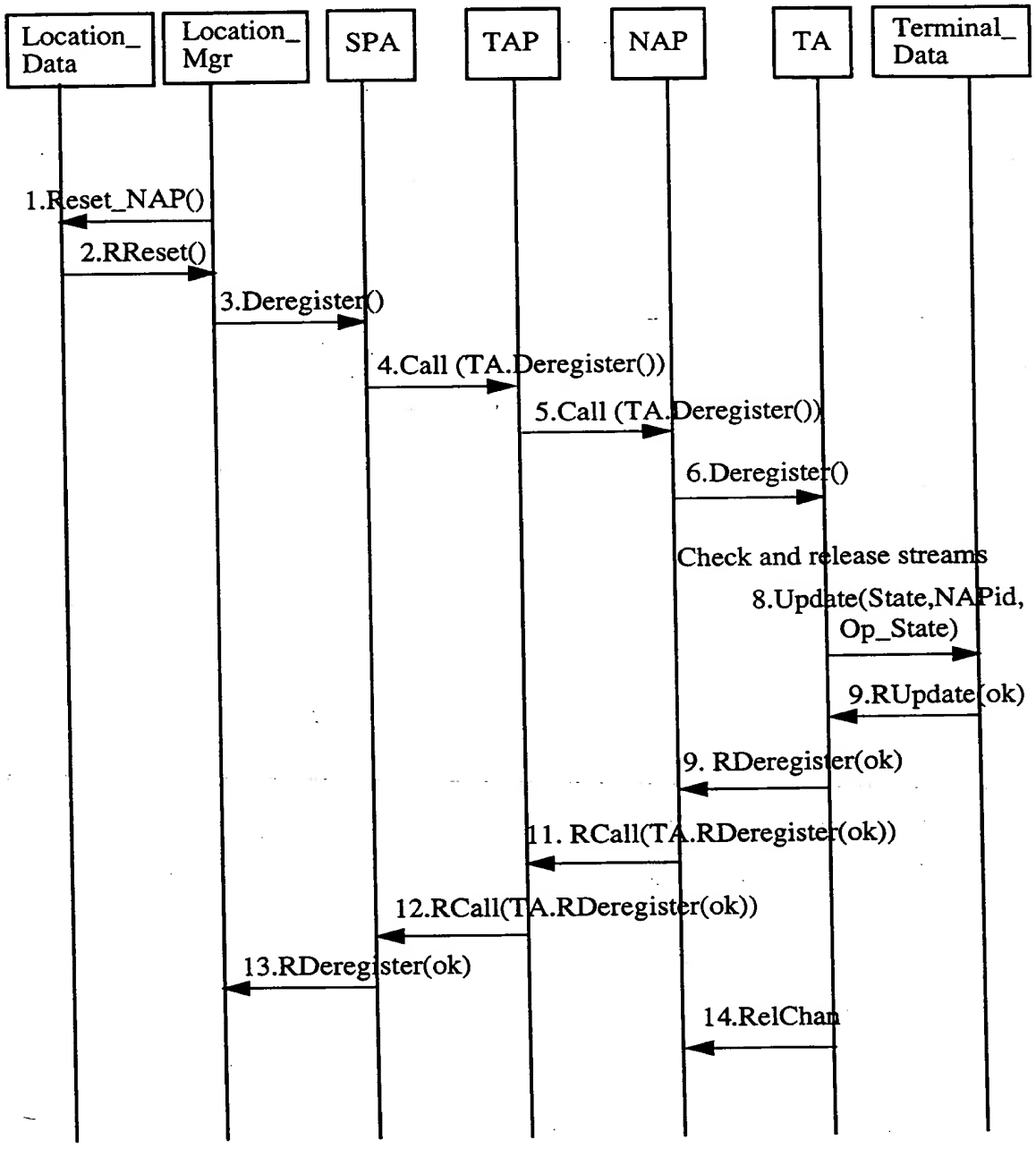


Figure 5.29 The procedure for the terminal deregistration

e. Disappearance

Disappearance is the second transition from the registered and confirmed State to the deregistered state. This transition occurs if all contact with the mobile DPE disappears and the pointers of the `Terminal_Data` are not all set to `Nil`. Since the Terminal_State is registered and confirmed, there must be activity, i.e there is at least one ongoing stream. When all contact has vanished, the `CSM` which is responsible for stream will be alerted by the other connection objects such as the `CC`. The `CSM` can then alert the `TA` by invoking the `OpChanLost()` on the `TA`. The `TA` proceeds with the updating of the `Terminal_Data`, i.e erases all entries in the `StreamTable`, sets the `Terminal_State` to Deregistered, the `NAPid` to `Void` and the `Operation_state` to "OFF".

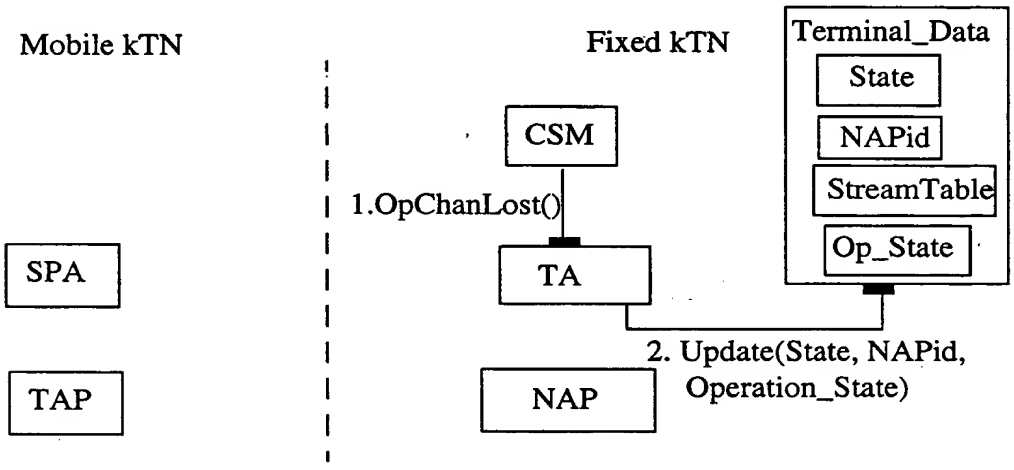


Figure 5.30 Disappearance - Second transition from the registered and confirmed state to the deregistered state

f. Activation failure

Activation failure is transition from the registered and unconfirmed state to the deregistered state. The transition occurs when an activity is initiated and fails because there is no response from the mobile DPE.

When an activity is requested by the mobile DPE and no contact can be established with the telecom system domain, it means that the terminal is already in the deregistered state and the `SPA` will reject the request. In fact, when the mobile DPE moves out of coverage and loses contact, the `Location_Mgr` will discover that it no longer receive some `NAP` identifier which is supposed to be broadcast periodically. This mechanism is used in GSM [GSM94a] and will be adopted in our design. The `Location_Mgr` will then set the `NAPid` in the `Location_Data` to `Nil`. The mobile DPE is hence in the deregistered state. It is worth noting that the fixed side or telecom system domain is not yet aware of this change of state. There is here a mismatch between the real state of the terminal and the one perceived by the telecom sys-

tem domain. Seen from the telecom system domain, the mobile DPE is still in the registered and unconfirmed until an activity is requested from the fixed DPE.

When an activity, which can be an operation or a stream, is requested form the fixed DPE, the **TA** will receive a `Call(CO1.OpY())` operation. It will then initiate the security procedures. The **NAP** will be ordered to get in touch with the mobile DPE and will fail. The security procedures will then also fail.

The **NAP** will reply to the **TA** with a `RSecurityProc(NoCon)` operation saying that there is no contact with the mobile DPE. The **TA** sets the `Terminal_State` to deregistered and the `NAPid` to `Nil`. It will also reply negatively to the requesting object by `RCall(NoCont)`.

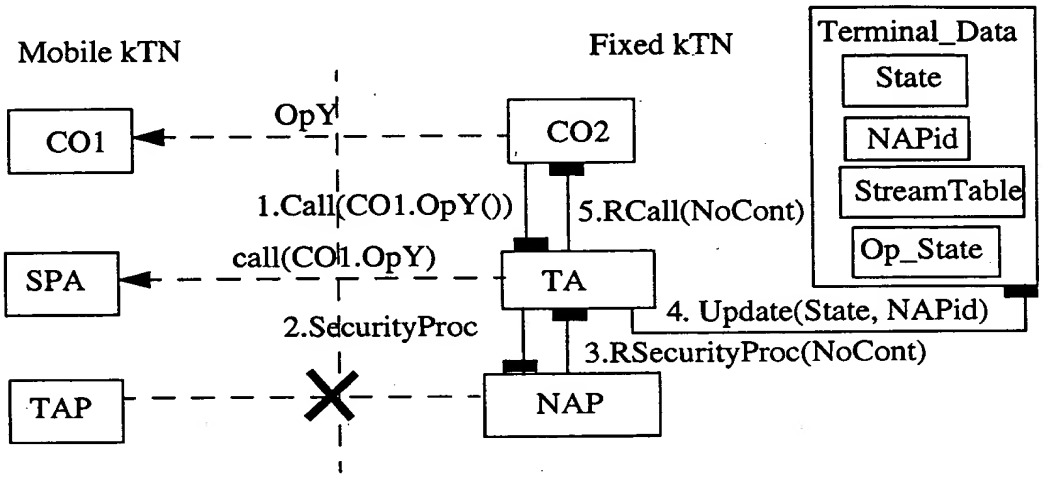


Figure 5.31 Activation failure - Transition from the registered and unconfirmed state to the deregistered state

The procedure for the activation failure is as follows:

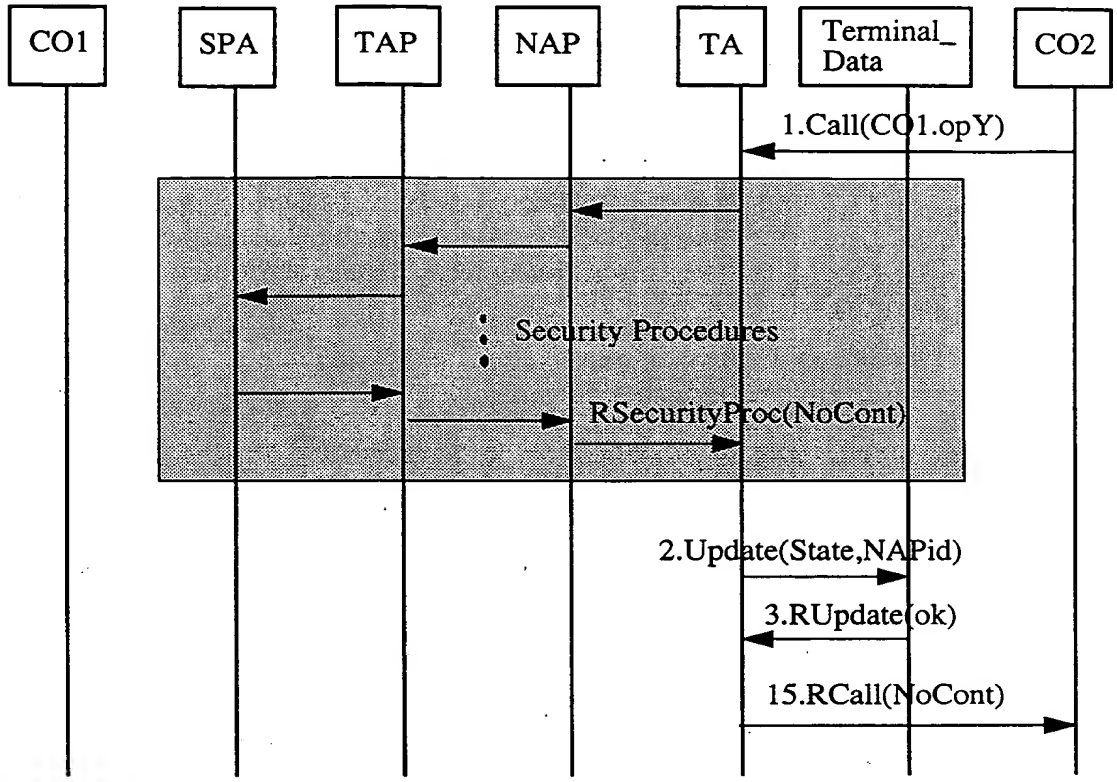


Figure 5.32 The procedure for the activation failure

g. Location updating

The location updating is the transition from one registered and unconfirmed state to another registered and unconfirmed state. The transition occurs when the mobile terminal moves from an area covered by one **NAP** to an area covered by another **NAP**. Periodically, the **NAP** is broadcasting its identifier which is read by the **TAP** and transferred to the **Location_Mgr**.

The **Location_Mgr** fetches the stored **NAP** identifier and compares it with the new one. Since the terminal has moved to another **NAP**, a mismatch occurs. The **Location_Mgr** updates the `NAPid` and invokes the operation `LocUpdate()` of the **SPA**. The **SPA** invokes `TAP.Call(TA.LocUpdate())`on the **TAP**. The **TAP** tries to connect itself to a **NAP**. If it succeeds, the terminal and the telecom system domain will proceed with the security procedures. If these procedures are successful, the **TAP** will invoke the operation `Call(TA.LocUpdate())` of the **NAP**. The **NAP** calls the operation `LocUpdate()` on the **TA**. The **TA** sets its NAP pointer to the new **NAP**. The **Location_Mgr** also sets its NAP pointer to the new **NAP**. The physical link between the **TAP** and **NAP** can then be disconnected. In the case that

one of the procedures fails, the terminal state will be changed to the deregistered state.

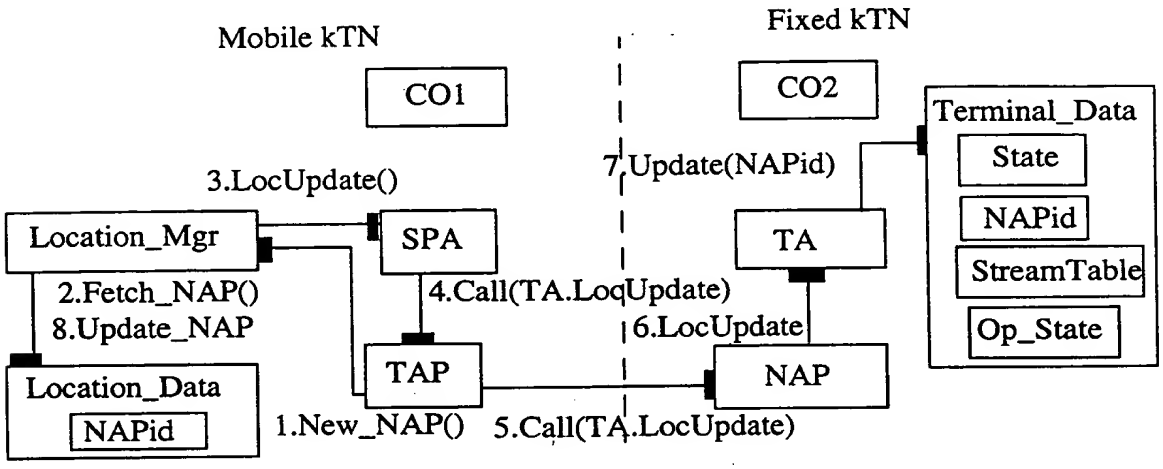


Figure 5.33 The location updating

The procedure for location updating is as follows:

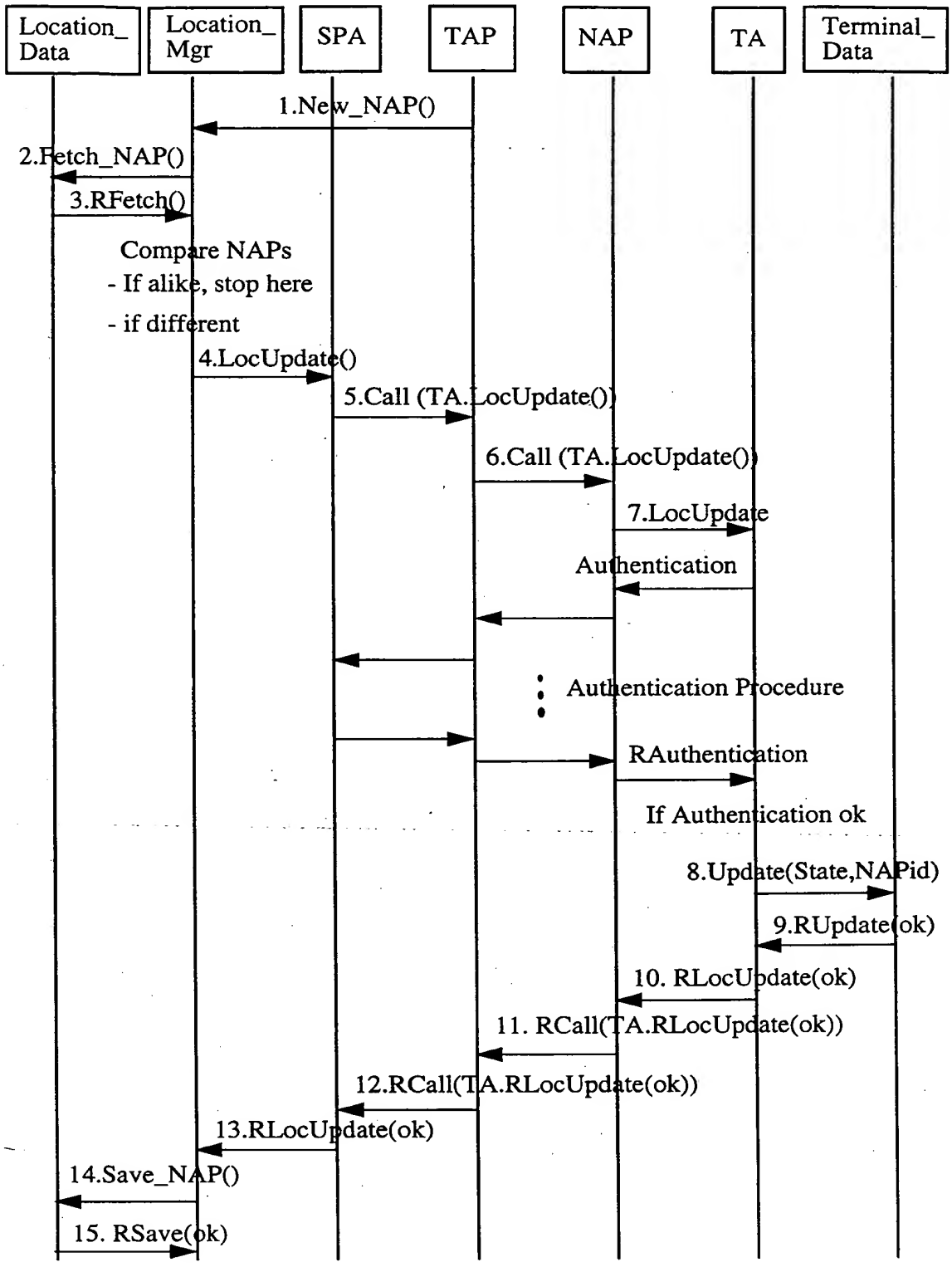


Figure 5.34 The procedure for the location updating

Before leaving this topic about the support of operations, there are two issues that need to be mentioned:

1. Overlapping of NAP coverage areas: In order to ensure operation continuity, there must be an overlapping between the NAP coverage area. Before leaving completely a NAP coverage area, the mobile DPE has already entered another NAP coverage area. At certain conditions, e.g quality of the channel operation, the mobile DPE will switch over to the new NAP.

2. Denial of service: It happens that the mobile DPE is not allowed to have access to the fixed DPE. The access control procedure for the terminal is one of the security procedures which be considered in details in Paragraph 7.3.

We have studied the functionality needed to enable operations between objects on the mobile terminal and objects on the telecom system domain. To support terminal mobility, it must also be possible to establish streams between the mobile DPE and fixed DPE. This will studied next.

5.3 ENABLING STREAM BINDING WITH THE MOBILE TERMINAL

We assume now that operational interactions can be established between the mobile terminal and the telecom system domain. From now on, in order to reduce the complexity and make the explanations easier, operations across the boundary between the mobile DPE and the fixed DPE, will be denoted as if they are directly addressed between peer-objects, although they still have to go through the objects **TA**, **NAP**, **TAP** and **SPA** as shown in Figure 5.35.

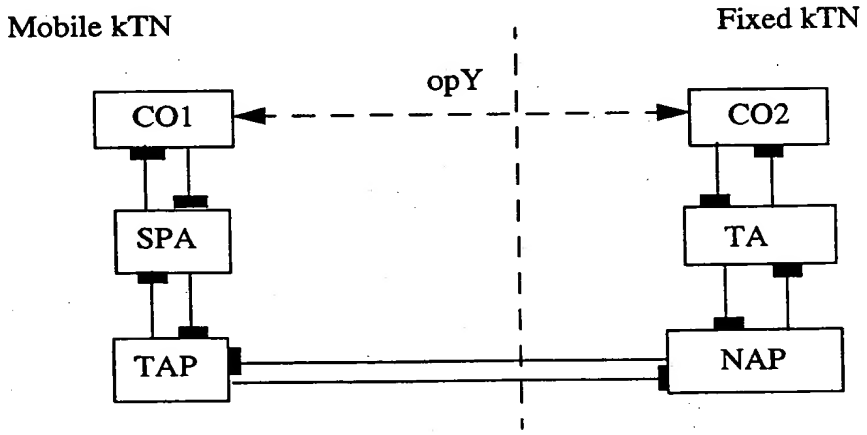


Figure 5.35 The convoy of operations between the mobile DPE and the fixed DPE

Consider the case of stream binding between the object CO_t residing on a mobile terminal t and the object CO_n residing on the telecom system domain. We will show how this can be done by use of an example.

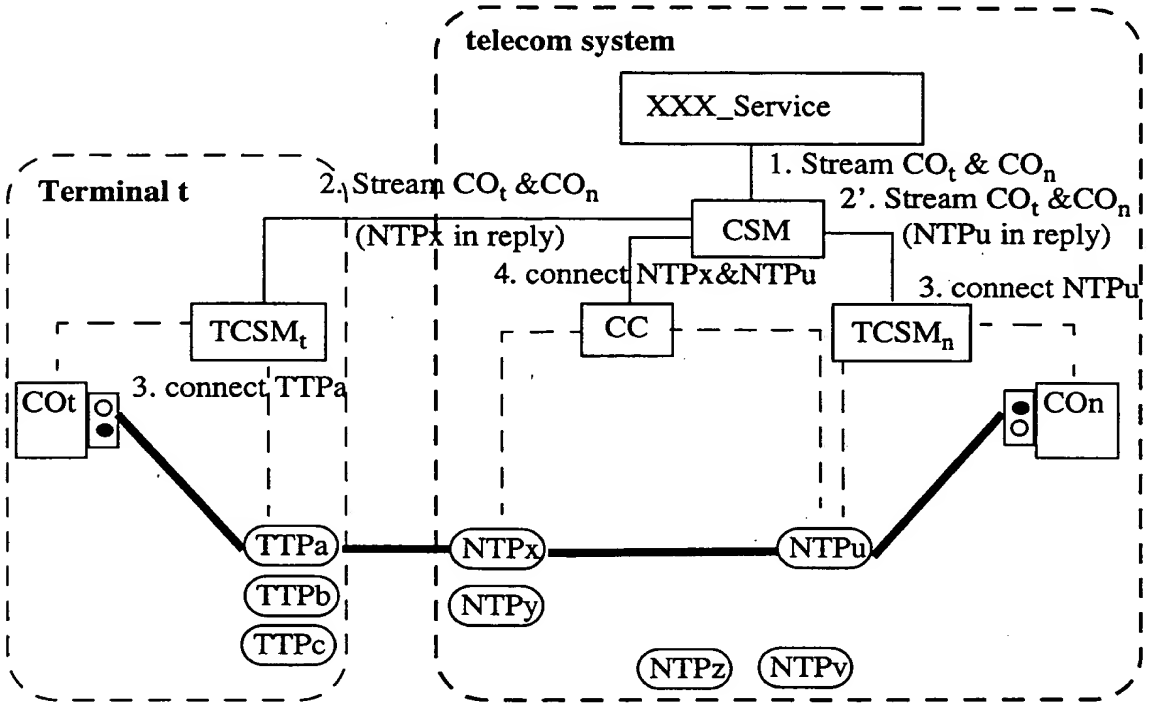


Figure 5.36 Stream establisment between two domains.

As explained in Chapter 4, streams are established between Terminal Termination Points (TTP) in the terminals and Network Termination Points (NTP) in the telecom system domain. The Terminal t has three **TTPa, TTPb** and **TTPc**. The telecom system domain has five **NTPx, NTPy, NTPz, NTPu** and **NTPv**. It is worth noting that the TTPs and NTPs may, but need not, be realized as computational objects. When the object `XXX_Service`, which is an arbitrary service, requests the `CSM` to establish a stream between CO_t and CO_n, the `CSM` will in turn request the $TCSM_t$ and $TCSM_n$ to establish local connection for CO_t and CO_n, respectively. This is referred as the Nodal Connection Graph in the TINA's Connection Management Architecture [TIN95b]. The $TCSM_n$ allocates **NTPu** for CO_n and returns the **NTPu** identifier to the `CSM`. Without mobility, the $TCSM_t$ would have allocated **TTPa** for CO_t and returned the **NTPx** identifier to the `CSM` since it knows that **TTPa** is connected with **NTPx**. The **CSM** will then request the **CC** to establish a stream between **NTPx** and **NTPu**.

The conditions for the success of the stream establishment are that **TTPa** is connected with **NTPx** and that the $TCSM_t$ knows about that. When the terminal t is moving, the **TTPa** will only be temporarily connected to **NTPx** and also with other **NTPs**, for example **NTPy**. We shall now study how the association of TTP to NTP can be set and modified when the terminal changes location.

The transport network used for stream flow can be modelled as network delimited by two or more NTPs (Network Termination Points) and capable of transporting bit stream between two NTPs. An NTP is the point at which a bit stream is accepted and/or delivered. Associated with an NTP is the characteristic of the stream accepted/delivered at the NTP such as frame structure, QoS, etc. Different NTPs may have different characteristics.

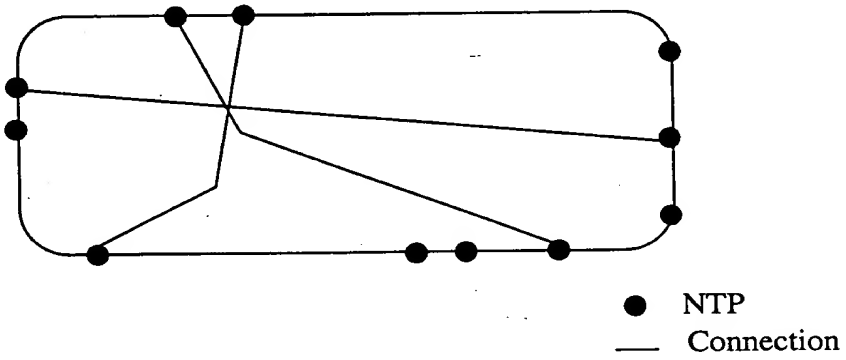


Figure 5.37 The transport network for stream flow

TINA has defined a similar model called the connectivity layer network [TIN96g]. However, the term Access Point is used instead of NTP. There is actually no difference between these two terms. The reason that the term NTP is used in this thesis is just to differentiate from the NAP (Network Access Point) which is the access point of the kTN (kernel Transport Network). It is very important to make a distinction between the kTN and the transport network for stream. As mentioned in [Aud96], these networks may be physically separate or share the same physical resources and represent only different concepts of the network. By separating the two network concepts, more freedom of choice for the kernel transport network is obtained and the evolution of the kernel transport network is made independent of the evolution of the technologies used for stream flow transport. Internet is a good example of this where the Web can be viewed as consisting of an interconnected set of application objects realised via a physical infrastructure (see Figure 5.38).

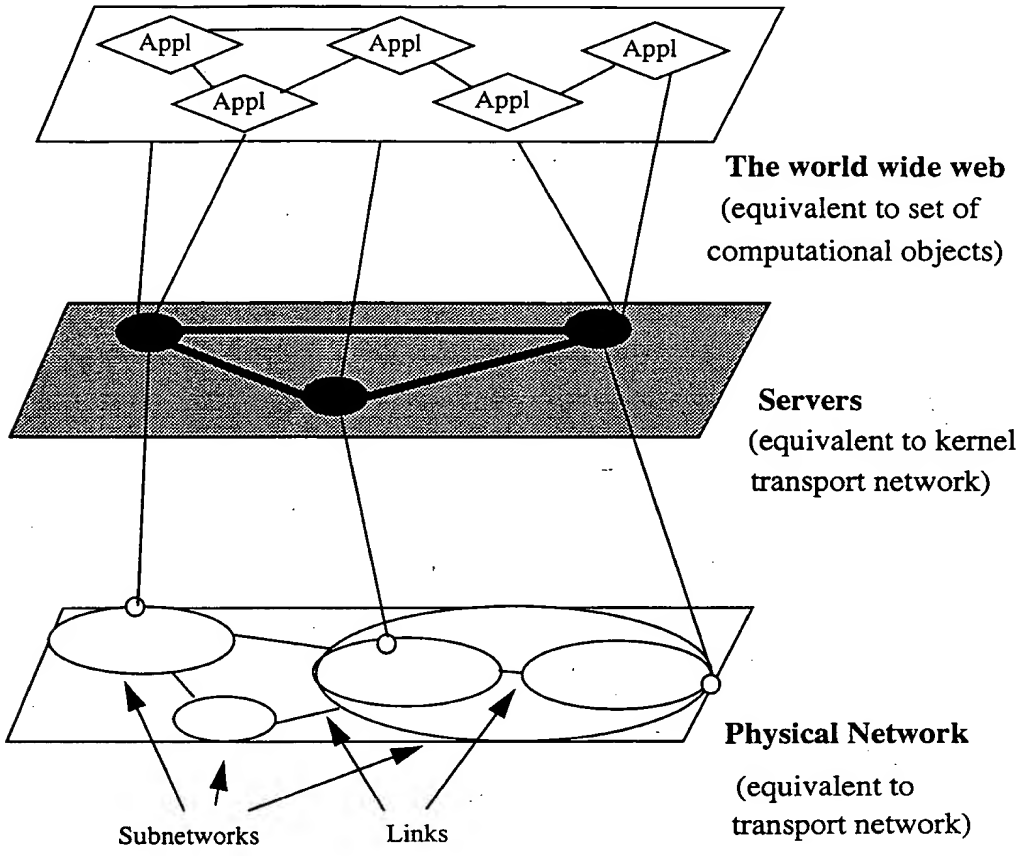


Figure 5.38 Internet as an example of the logical distinction between the kernel transport network and the transport network

The NTPs can be grouped into an NTPPool for example according to their geographical location. The area covered by the telecom system can hence be divided into smaller areas, each one served by one NTPPool.

As shown in Figure 5.39, when the mobile terminal is in area A, a stream flow between an object residing in the terminal and another object residing in the telecom system goes through an NTPa belonging to the $NTPPool_a$. When the terminal leaves the area A and enters, for example area B, then its stream flows must go through NTPb belonging to $NTPPool_b$. The $TCSM_t$ must somehow be aware that the terminal has moved and that another NTP belonging to another NTPPool should be used.

For each NTPPool, let us define an object called `NTP_Manager`. The `NTP_Manager` is in charge of an area and has information about all NTPs in that area, i.e which NTPs are available and their characteristics (QoS parameters). The `NTP_Manager` is periodically broadcasting its identifier in its area. The

NTP_Manager identifier will be used by the $TCSM_t$ when it wants to communicate with the NTP_Manager.

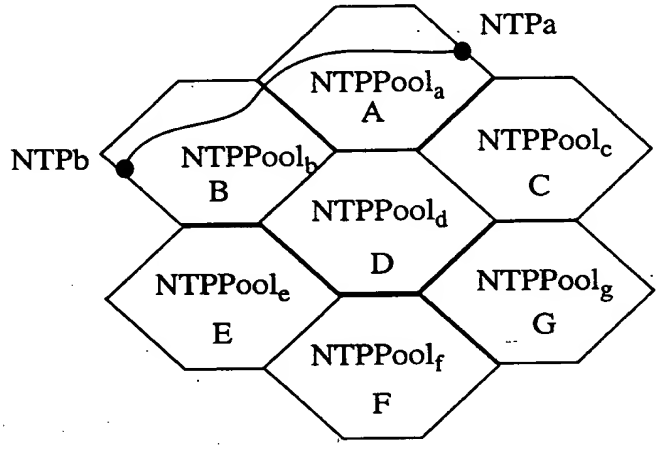


Figure 5.39 The coverage area divided into area served by NTPPools

When the terminal t is entering area A, the $TCSM_t$ will get the identifier of the $NTP\_Manager_A$, which is received through the TAP. Note that the kTN is used for the broadcast of NTP information. Upon connection request from the CSM, the $TCSM_t$ will ask the NTP_Manager to allocate an NTP for its TTP. The $TCSM_t$ will return the identity of this NTP back to the CSM. The CSM can then request the CC to establish a connection between this NTP to be used for CO_t object and the NTP to be used for CO_n.

Actually, the TCSM of the fixed node closest to the area A may be in charge of the allocation of NTPs. However, its is better to place all the connectivity functions which are specially related to terminal mobility in a dedicate object such as the NTP_Manager which can be modified without any consequence for the TCSM. Furthermore, according to TINA [TIN95b], a TCSM is defined for a node in the transport network and the notion of coverage area presented here has nothing to do with a node. A coverage area may or may not correspond to a node. A node may in fact contain several areas when it has numerous NTPs covering a large geographical area which need to be grouped into smaller areas.

The area covered by the NTP_Manager may or may not correspond to the area covered by the NAP since the two objects may be responsible for the connectivity with two separate physical networks, the kernel transport network and the transport network supporting streams.

There are three correspondence alternatives:

1. a NAP's area corresponds to a NTP_Manager's area.
2. a NAP's area corresponds to several NTP_Manager's areas.
3. an NTP_Manager's area corresponds to several NAP's areas.

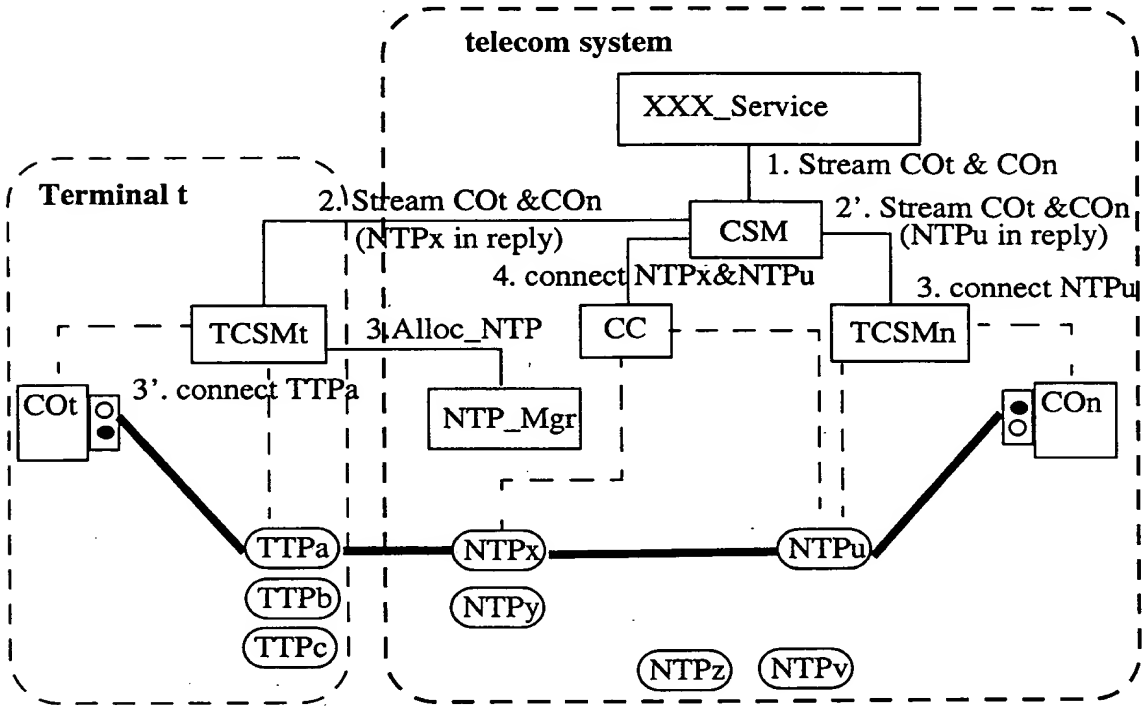


Figure 5.40 Allocation of NTP for TTP

Alternative 1

The coverage area of the NTP_Manager is the same as the one of the NAP. When the mobile terminal moves from one area to another, a location updating is executed. The terminal is changing from one registered state to another. If the terminal is in the registered and confirmed State, i.e there is activity going on, either operation or stream, then in addition to a location updating, a handover will be performed.

Alternative 2

Several NTP coverage areas constitute a NAP coverage area. When the mobile terminal moves from one NTP area to another, where the NTP areas belong to the same NAP, then a location updating is not required. There is no change of the terminal state. If the terminal is in the registered and confirmed State, i.e there is a stream

flow going on, then only a handover is required. If the terminal is in the registered and unconfirmed State, no action is needed.

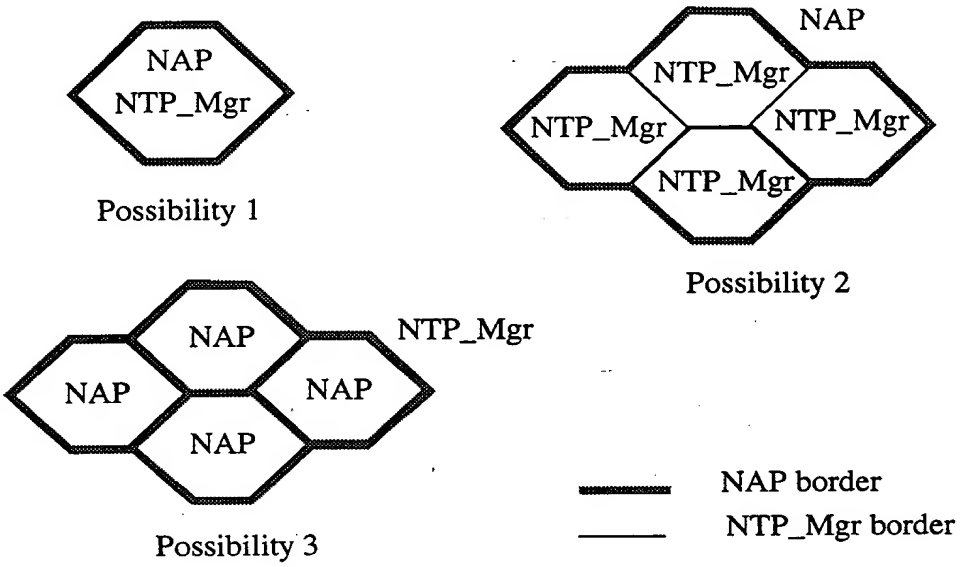


Figure 5.41 Correspondence possibilities between the NAP and the NTP_Mgr

Alternative 3

An NTP coverage area consists of several NAP coverage areas. When the mobile terminal changes from one NAP area to another where both belong to the same NTP area, a location updating is necessary since the terminal changes form one registered state to another. When the terminal changes NTP area and the terminal is in the registered and unconfirmed State, i.e there is no stream flow going on, only a location updating is required. If the terminal is in the registered and confirmed State, i.e there is an outgoing stream flow, both a location updating and a handover will be performed.

The choice of the appropriate correspondence alternative for a particular system is a matter of dimensioning and configuration and we shall not study it further here.

5.3.1 Handover

When the mobile DPE moves from one NTP coverage area to another, there are two ways to support the availability of service:

- **Continuous terminal mobility** ensures the continuity of service by using the continuous handover mechanism. GSM is an example of this type of terminal mobility [GSM94c].
- **Discrete terminal mobility** ensures only the continuity of the service ses-

sions but not the continuity of service, i.e the service sessions are suspended before and during the transition and resumed when the mobile DPE has reached the new NTP coverage area. This is referred as discrete handover or session mobility.

We shall now consider successively both cases.

5.3.2 Continuous handover

5.3.2.1 Examination of the situation

To ensure that stream flows are not disrupted when the mobile terminal is moving from one NTP coverage area to another, there must be an overlap between coverage areas. In the intersection area $A \cap B$ of to NTP coverage areas A and B, the mobile terminal has contact with both `NTP_Managers` and stream flow can be established with NTPs belonging to both NTPPools.

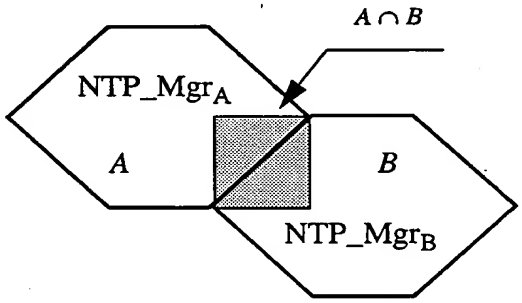


Figure 5.42 Overlap between NTP coverage areas

Suppose that the mobile terminal is originally in area A and there is a stream flow between its object CO_t and another object CO_n in the telecom system domain. The stream goes through TTP_a, NTP_{Ax} and NTP_u. When the terminal enters the intersection area $A \cap B$, the simplest and most obvious way to ensure that there is always a stream flow between CO_t and CO_n, is to establish an additional stream going through an NTP_{BX} belonging to the $NTPPool_B$ as shown in Figure 5.43. The object CO_t is bound to a TTP_b which is connected to NTP_{BX}. NTP_{BX} is connected to NTPv which is bounded to CO_n. When the mobile terminal is in the intersection area $A \cap B$, there are two identical streams between CO_t and CO_n. When the mobile terminal leaves area A completely, the original stream going through NTP_{AX} can be released.

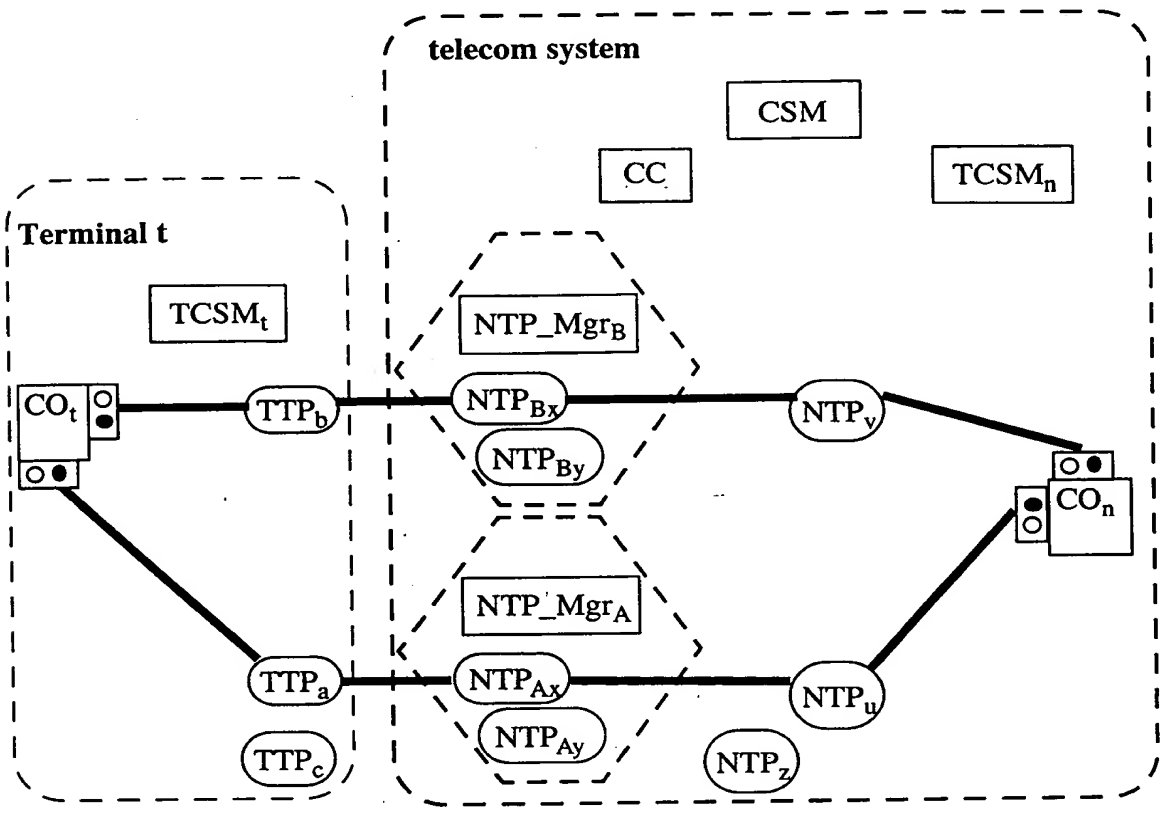


Figure 5.43 Establishment of an additional stream flow

There are, however, several problems with this simple scheme. In the intersection area $A \cap B$, the source object, which in our example is CO_t, has to duplicate the information and send it to both streams. The sink object, which in our example is CO_n has to read from both streams, compare the contents and discard the duplicated information. Both the source object and the sink object have to be aware that either itsself or its peer-object resides on a mobile DPE node. They must also have built-in functionality to handle handover. Another problem occurs when the overlap of the coverage areas is considerable or when the mobile DPE stays in the overlap area for a long time. Then two streams are established for unjustified reason using more resources than necessary. Improvements of this simple scheme are required.

5.3.2.2 Handover procedure

A handover procedure consists of the following steps:

1. To detect that handover may be necessary. This can be done via the detection that the mobile DPE is in an intersection area such as $A \cap B$ or $A \cap B \cap C$, etc.
2. To decide that handover must be executed. The decision can be taken based

on different criteria such as transmission quality, distance between the mobile DPE and the NTP, etc. The criteria are system dependent and cannot be implemented in a generic way. Furthermore, the decision for handover can be made by the mobile DPE or by the telecom system domain or also by both.

3. To perform the handover in a fastest and most secure way.

4. To move to a new stable state.

5.3.2.3 Necessary computational objects

In order to realise the procedure described above, several new objects are required in the GMS.

First, an object called **Handover_Initiator** is introduced to decide and supervise the handover procedure. The **Handover_Initiator** can be created by the **TCSM** on the mobile DPE if the handover decision should be made by the mobile terminal. It can also be created by the **NTP_Manager** on the fixed DPE if the handover decision should be made by the telecom system domain. The **Handover_Initiator** object encapsulates the functions necessary to realise a specific decision criterion. Hence, for a specific system using a specific criterion, a specific version of the **Handover_Initator** is needed. All the **Handover_Initiator** objects have, however, the same operational interface toward other objects in the system.

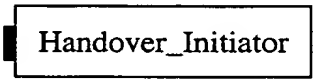
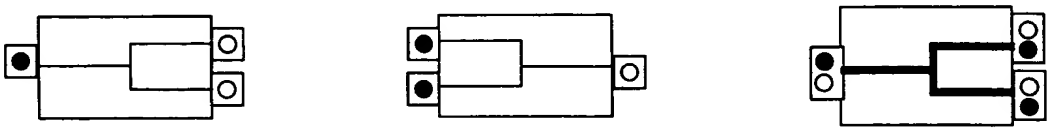


Figure 5.44 The Handover_Initiator object

Second, two objects called **Handover_Source** and **Handover_Sink** are introduced. Each object has at least three stream interfaces. The **Handover_Source** object has the ability to connect an input stream interface with two output interfaces and to duplicate the information flow form the input stream interface. The **Handover_Sink** object has the ability to connect two input stream interfaces into one output stream interface and to discard the duplicated information from two identical input streams. In the case where there is a bidirectional stream flow between two objects, it is more convenient to use an object called **Handover** having the functionality of both the **Handover_Source** and **Handover_Sink** as shown in Figure 5.45. The binding of stream interfaces belonging to the same object is described in [Do96a].

The **Handover_Source** should be introduced after the source object using the stream direction as reference and the **Handover_Sink** object should be introduced before the sink object.



a. Handover_Source b. Handover_sink c. Handover

Figure 5.45 The Handover objects

It is worth noting that the **Handover** objects represent in fact a combination of both hardware and software modules which are specific to a system. Although these objects may have different internal functions and mechanisms they will have the same external interfaces.

5.3.2.4 Example of handover

Let us consider the same example as in Paragraph 5.3.2.1. We introduce now two objects $\mathbf{Handover_t}$ having three stream interfaces St1, St2, St3 and $\mathbf{Handover_n}$ having three stream interfaces Sn1, Sn2 and Sn3. We let the mobile DPE make the handover decision through a **Handover_Initiator** object residing in the mobile DPE.

a. Before handover

When the **CSM** receives the request to bind $\mathbf{CO_t}$ and $\mathbf{CO_n}$, it knows that $\mathbf{CO_t}$ is residing on a mobile DPE by reading the name of $\mathbf{CO_t}$. It may then deduce that a handover object is necessary. It creates or requests a Service Factory object to create an object $\mathbf{Handover_n}$ and proceeds with the procedure to establish a stream between $\mathbf{CO_n}$ and the stream interface Sn1 of $\mathbf{Handover_n}$. This is done without problems since both $\mathbf{CO_t}$ and $\mathbf{Handover_n}$ are residing in the fixed kTN. The CSM requests $\mathbf{Handover_n}$ to bind internally the stream interfaces Sn1 and Sn2.

Towards the mobile side, the **CSM** requests the $\mathbf{TCSM_t}$ to establish a local connection for $\mathbf{CO_t}$. The $\mathbf{TCSM_t}$ knows that it is residing on a mobile DPE and that a handover object is necessary. It creates or requests a Service Factory object to create an object $\mathbf{Handover_t}$. It will then establish a stream between $\mathbf{CO_t}$ and stream interface St1 of $\mathbf{Handover_t}$. The $\mathbf{TCSM_t}$ will then ask the $\mathbf{Handover_t}$ to bind internally the stream interfaces St1 and St2. The $\mathbf{TCSM_t}$ requests also the $\mathbf{Handover_t}$ to bind St2 to TTP_a. The $\mathbf{TCSM_t}$ returns the corresponding NTP identifier to which TTP_a is connected, namely NTP_{AX}, to the **CSM**.

The **CSM** then connects the $\mathbf{Handover_n}$ object to NTP_{AX}. When this is accomplished, a stream is established between $\mathbf{CO_t}$ and $\mathbf{CO_n}$.

When the terminal moves into the intersection area $A \cap B$, the $\mathtt{TCSM_t}$ will receive the identifier of the `NTP_MgrB` and know that a handover may be necessary. It creates or requests a Service Factory to create a `Handover_Initiator` object in the mobile DPE.

The `Handover_Initiator` encapsulates the mechanism for handover decision and starts to operate immediately after its creation.

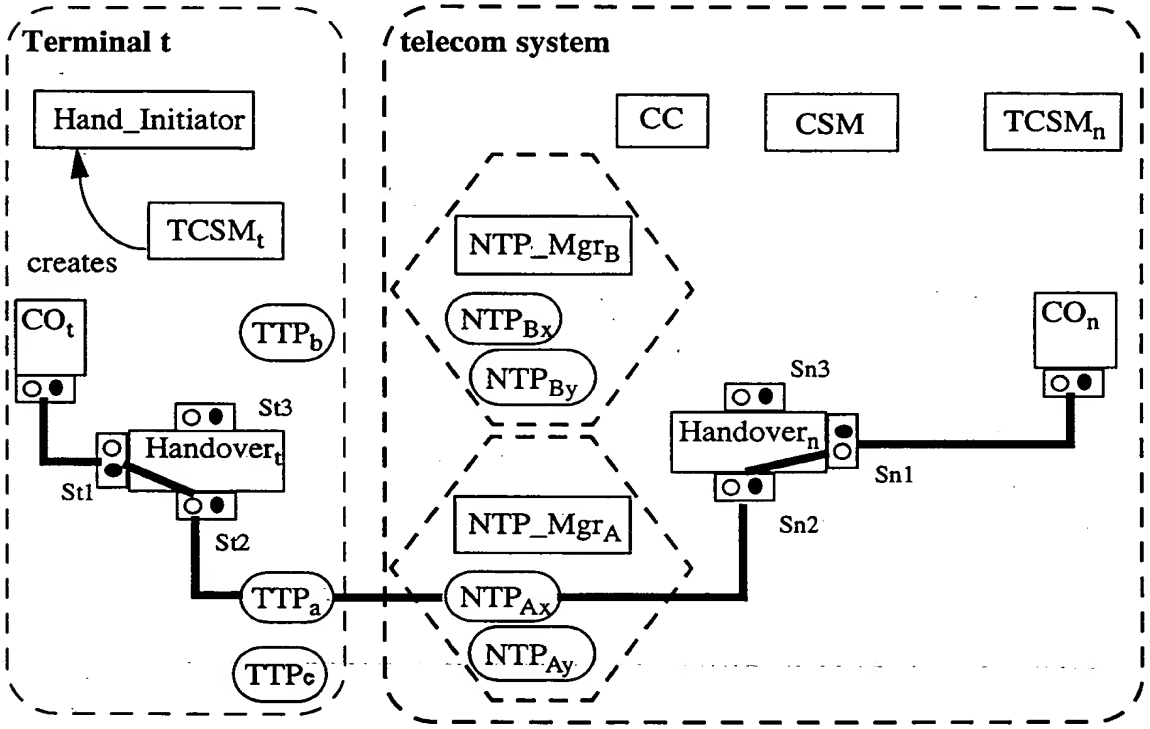


Figure 5.46 The stream flow before handover

b. During handover

When the criterion for handover is fulfilled, the `Handover_Initiator` will start the handover procedure. It requests the `CSM` to establish a stream between the interface St3 of $\mathtt{Handover_t}$ and the interface Sn3 of $\mathtt{Handover_n}$. The `CSM` requests the $\mathtt{TCSM_t}$ to bind St3 to a TTP. The $\mathtt{TCSM_t}$ binds St3 to TTP_b. The $\mathtt{TCSM_t}$ returns the corresponding NTP identifier to which TTP_b is connected, namely NTP_{BX}, to the `CSM`. The `CSM` then connects the $\mathtt{Handover_n}$ object to NTP_{BX}.

Everything has now been prepared for the switchover. The `Handover_Initiator` requests the $\mathtt{Handover_t}$ to bind internally the interfaces St1 and St3, and the

$\texttt{Handover}_n$ to bind internally the interfaces Sn1 and Sn3. The `Handover_Initiator` then requests the `CSM` to release the stream between the interfaces St2 of $\texttt{Handover}_t$ object and Sn2 of $\texttt{Handover}_n$ object. It also requests the $\texttt{Handover}_t$ object to unbind internally St1 and St2 and the $\texttt{Handover}_n$ to unbind Sn1 and Sn2.

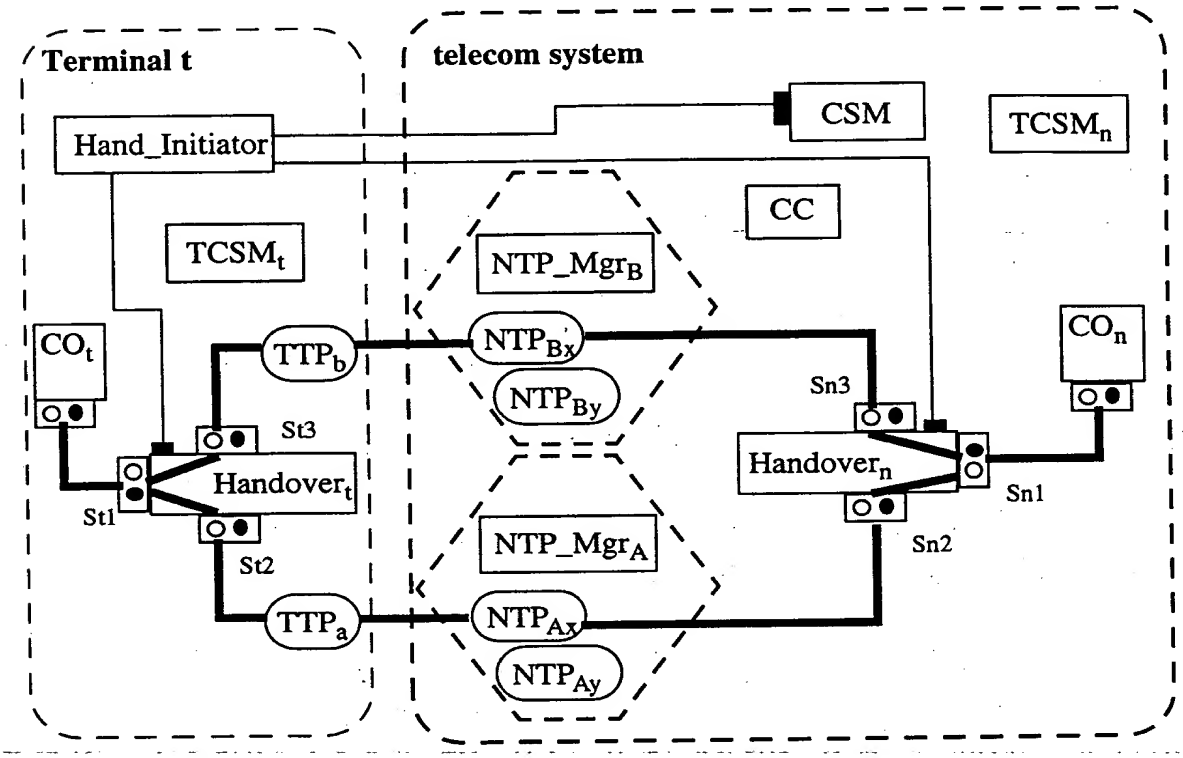


Figure 5.47 The stream flow during handover

In Figure 5.47 we just show that there are interactions between the `Handover_Initiator` and the `CSM`, and the $\texttt{Handover}_t$ and the $\texttt{Handover}_n$ without any further specification in order to preserve clarity.

c. After Handover

As shown in Figure 5.43, after the accomplishment of the handover procedure the `Handover_Initiator` object will terminate itself. The stream between the objects $\texttt{CO}_t$ and $\texttt{CO}_n$ follows now the new path without disruption.

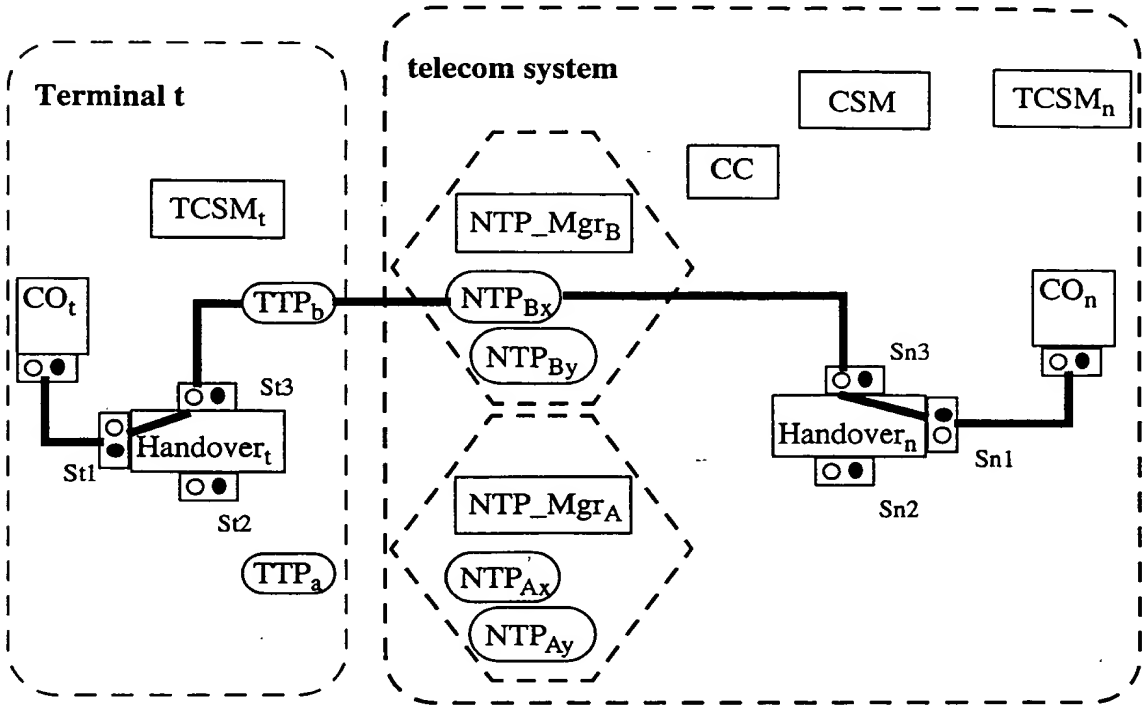


Figure 5.48 The stream flow after handover

5.3.3 Discrete handover

Without overlap between the NTP coverage areas, the mobile DPE will loose all contact with the telecom system domain when it moves out of an NTP coverage area. The continuity of service cannot be ensured. However, the service sessions in use can be suspended in order to be resumed later when the mobile DPE arrives at the new NTP coverage area. In a way, the service sessions can be considered as non-disrupted. The continuity of sessions requires the intervention of the user, i.e the user must explicitly order the suspension of the sessions in use before moving to another NTP area. He must also explicitly resume these sessions later on. The continuity of sessions is also referred as session mobility and shall be treated in details in Paragraph 9.6.

5.4 CONCLUSION

In this chapter we have studied how terminal mobility can be supported. To support mobility, the GMS must have the following computational objects:

a. To support operational interactions:

- In the mobile DPE:
  - SPA
  - TAP
  - Location_Mgr
  - Location_Data
- In the telecom system:
  - TA
  - NAP
  - Terminal_Data

b. To support stream flows:

- In the mobile DPE:
  - Handover objects
  - Handover_Initiator
- In the telecom system
  - NTP_Manager
  - Handover objects
  - Handover_Initiator

In addition, the following objects defined by the TINA Connection Management need to have enhanced functionality in order to cooperate with the GMS objects:

- CSM
- TCSM

These two objects must interact with the GMS objects such as the TA, the SPA, etc. in order to support streams. They must have the ability to distinguish between objects residing on a mobile DPE and objects residing on the fixed DPE in order to decide the creation of Handover objects for streams spanning the terminal domain and the telecom system domain. The TCSM must also have the ability to detect that a handover may be necessary in order to create the Handover_Initiator object which controls the handover procedure.

6

Supporting user mobility

6.1 INTRODUCTION

In the last chapter we have only considered terminal mobility. No distinction has been made between the user and the terminal. Seen from the telecom system domain, it is not relevant to know who or which is using the terminal but only the fact that the terminal has been used to access the telecom system domain. The user is completely assimilated to the terminal. In this chapter we shall enhance mobility by introducing user mobility. By user mobility it is meant that the user is allowed to access the telecom system using any terminal and at any network access point. Note that the user can be a person, a software process or another terminal.

We shall study and find all the functions which are necessary to support user mobility. The related computational objects will be introduced and described gradually as the needs arise and in a natural way.

6.2 DISTINCTION BETWEEN THE USER AND THE TERMINAL

In order to offer user mobility i.e to allow the user to move and access the telecom system independently of both the terminal and the network access point, it is necessary to make a distinction between the user and the terminal. However, the user can still be a person, a software process or a hardware device. A particular case arises if the user is a mobile DPE node. Such cases of "double" terminal mobility may arise: a train or ferry offers DECT cells which are connected to the fixed network by GSM. However such a "double" terminal mobility does require other mechanisms than those described in Chapter 5.

As explained in Chapter 4, we have a system consisting of three domains: the telecom system domain, the terminal domain and the user domain.

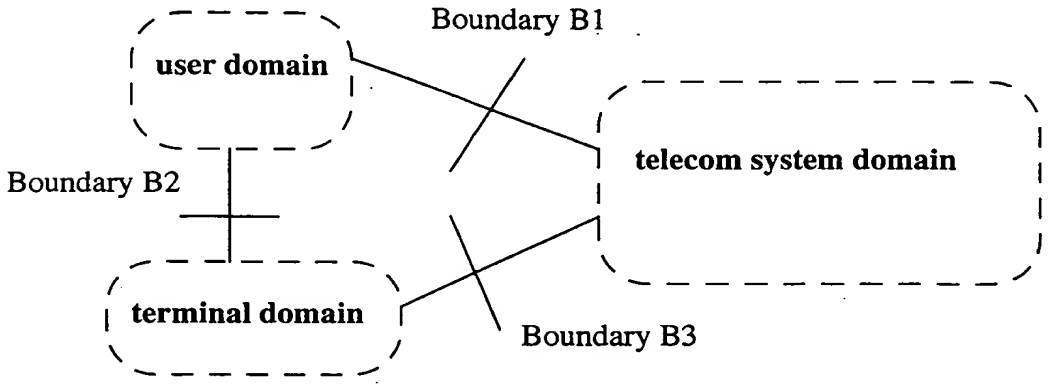


Figure 6.1 A system consisting of three domains

Actually, the telecom system domain is a composite domain consisting of several administrative and technology domains as explained in Chapter 4. Between the user domain and the terminal domain, there are both an administrative boundary and a technology boundary. The technology domain is also referred as the perceptual reference point [TIN95e]. Between the terminal domain and the telecom system domain, there are both an administrative boundary and a technology boundary. These two boundaries were studied in Chapter 5 where the support of terminal mobility was handled. Between the user domain and the telecom system domain, in addition to a technology boundary there is an administrative boundary since the user and the telecom system domain belong to different administrations or stakeholders. It is worth noting that the boundary B1 is an "indirect" boundary between the user domain and the telecom system domain since the user never interacts directly with the telecom system but always indirectly through a terminal domain. In other words, the boundary B1 is realized through boundary B2 and boundary B3. However, the boundary B1, as a logical concept, is essential for user mobility and will be studied in details. The support for user mobility is indeed to make the boundary B1 appear direct seen from the user. Although very important, the boundary B2 will be studied only briefly and only operations related to user mobility will be described since the man-machine interface problem falls beyond the scope of this thesis.

6.3 MORE SPECIFIC DEFINITION OF USER MOBILITY

Let us now consider an example consisting of a telecom $system_N$ domain, a $Terminal_m$ domain and a $User_a$ domain. The $User_a$ domain consists of the $\texttt{User}_a$ object

and an arbitrary computational object called CO1. Note that the object CO1 may or may not reside physically in the $terminal_m$. In the case where the user is also a terminal or hardware device, CO1 may reside on the user itself and not on the $Terminal_m$ domain.

To offer user mobility means to allow the whole user domain including CO1, to move and change terminal and still be able to interact with an object CO2 residing in the telecom $system_N$ domain. By interactions it is meant both operations and stream flows.

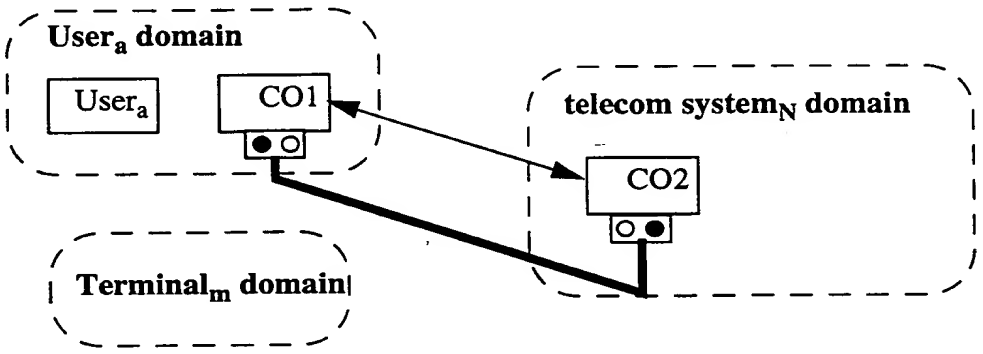


Figure 6.2 User mobility defined in terms of interactions

Let us now introduce the interceptors necessary to enable interactions between domains. At the boundary between the $User_a$ domain and the telecom $system_N$ domain an interceptor called PD_UA_a ($ProviderDomain\_User\_Agent_a$) is introduced in the telecom $system_N$ domain. This interceptor represents the $User_a$ in the telecom $system_N$ domain and assumes security functions such as identification, authentication, encryption, etc. We will see later that the interceptor PD_UA_a is also responsible for the functions supporting the mobility of the user. At the boundary between the $User_a$ domain and the $Terminal_m$ domain an interceptor called TD_UA_a ($TerminalDomain\_User\_Agent_a$) is introduced in the $Terminal_m$ domain to represent the $User_a$ and to assume the security functions of the $Terminal_m$.

We assume further that every terminal must have an owner and it is reasonable to define this owner as the default user of the terminal. A legal and operative terminal will always have a default user. This default user is permanently registered at the terminal. Definition of a default user for a terminal allows us to implement access control procedures which may restrict the usage of the terminal. Of course, the default user can also be registered at another terminal but the former registration is always valid and we have a multiple terminal registration case. There is therefore introduced in the $Terminal_m$ domain an interceptor TD_UA_m representing the default user. Correspondingly, an interceptor PD_UA_m is introduced in the telecom $system_N$ domain.

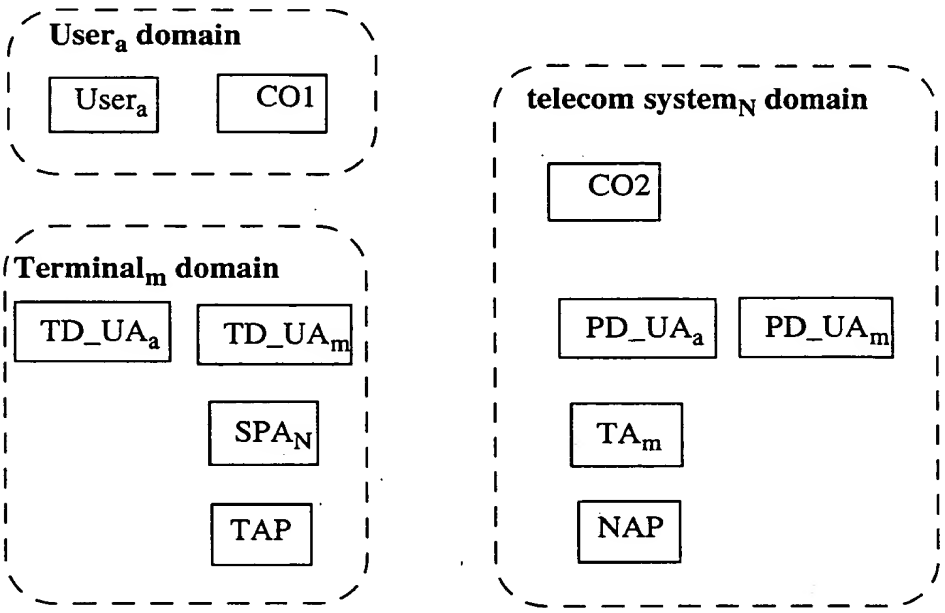


Figure 6.3 Computational model with necessary interceptors

In Figure 6.3, only the objects defined and specified in the previous chapter which are necessary to our coming discussion, such as SPA_N, TAP, TA, etc. are shown. In the succeeding paragraphs, we shall study how operations and stream flows can be enabled between the user domain and the telecom system domain while the user domain is moving and changing between different terminal domains. We shall identify the necessary functions and introduce gradually the computational objects assuming these functions.

6.4 ENABLING OPERATIONS BETWEEN THE MOBILE USER AND THE TELECOM SYSTEM

Since operations can be initiated either by the $User_a$ or by objects the telecom system$_N$ domain, we will consider the two cases separately.

6.4.1 Operations initiated by the mobile user

Suppose now that CO1 wants to invoke an operation OpY() on the object CO2. The invocation is not done directly.

First, an operation Call(CO2.OpY()) is issued to the object TD_UA_a. This shows that there must be an association between CO2 and TD_UA_a. The definition of this as-

sociation will be discussed later. The TD_UA_a will then invoke the operation `Call(O2.OpY())` on its peer-object PD_UA_a. What TD_UA_a needs in order to issue the operation request is the identifier of the object PD_UA_a. Since both TD_UA_a and PD_UA_a represent the same user and there is only one unique PD_UA_a, it is not a problem for TD_UA_a to find the identifier of PD_UA_a. Indeed, both of these two objects hold the identity of the user whom they represent.

If terminal mobility is supported in the system then, according to the results obtained in Chapter 5, it is possible for the TD_UA_a object to interact with the PD_UA_a by using mechanisms defined there. The operation `Call(O2.OpY())` on PD_UA_a is converted to an operation `Call(`PD_UA_a`.Call(CO2.OpY()))` of the SPA_N. This conversion is possible because, as in Chapter 5, a call to PD_UA_a, an object residing on the fixed kTN part, must always go through the SPA_N.

From this step until step 6 in Figure 6.3, all the succeeding operations are related to the terminal mobility support. In Figure 6.3 we show all the operations just, also those belonging to terminal mobility support, for completeness and clarity. The SPA_N issues a operation request `Call(`PD_UA_a`.Call(CO2.OpY()))` on its peer-object TA_m. This request is converted to the operation `Call(`TA_m`.Call(`PD_UA_a`.Call(CO2.OpY())))` on the TAP. The TAP will then forward this operation to the NAP to which it is currently connected with. The NAP will then issue an operation request `Call(`PD_UA_a`.Call(CO2.OpY()))` to the TA_m. The TA_m will then invoke `Call(CO2.OpY())` of the PD_UA_a. The PD_UA_a will now call the operation `OpY()` of the object CO2. This concludes the conveyance of the operation `OpY()`.

The invocation of operations by objects in the user domain are seamlessly supported. The only information required by the user domain is that CO2 is residing on the fixed kTN (or located in the telecom system domain). No additional information is required.

It is worth noting that this conclusion agrees with the specifications of UPT (Universal Personal Telecommunications [ITU94] [CCI92c] [AJ92] [Eri94] which offers personal mobility for telephony. For outgoing calls, the user does not really need to make a registration. He/she has only to dial the UPT access code, enter his/her UPT number and PIN code for authentication and make a telephone call.

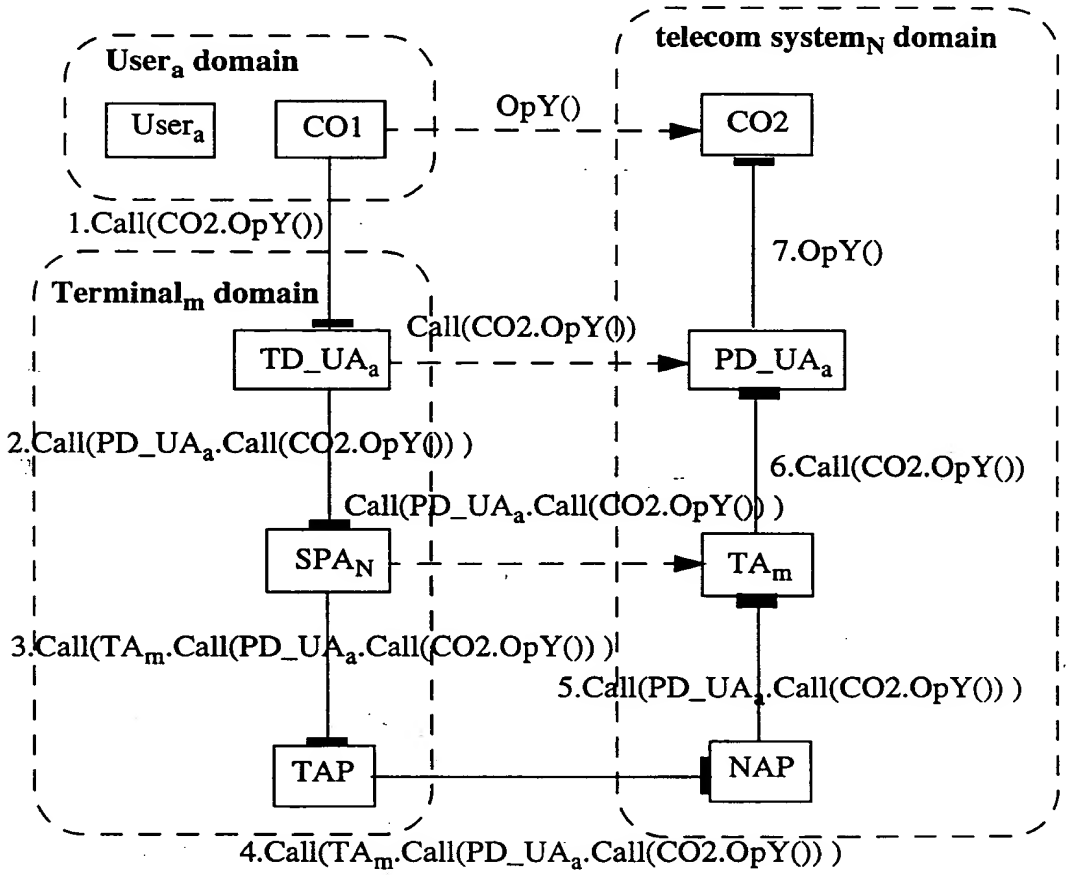


Figure 6.4 Operations initiated by the mobile user

6.4.2 Operations initiated by an object in the telecom system domain

Suppose now that an object `CO2` in the telecom system domain wants to invoke the operation `OpX()` of `CO1`. This operation will be converted to the operation `Call(CO1.OpX())` of the `PD_UA`$_a$. In order to make the conversion possible the information that `CO1` belongs to the User$_a$ domain and not to any other user domain, must be available in the telecom system domain. This problem will be treated in Chapter 8. Until then we simply assume that this information is available.

The `PD_UA`$_a$ will then invoke the operation `Call(CO1.OpX())` on the object `TD_UA`$_a$. The first requirement is to find the identifier of `TD_UA`$_a$. For the time being we just suppose that such information is available. We shall soon come back to how it can be found. The operation request is thereafter converted to the operation `Call(TD_UA`$_a$`.Call(CO1.OpX()))` on the `TA`$_m$. To be able to do that, the

PD_UA_a must have the identifier of TA_m or the identity of the terminal, namely $Terminal_m$ where the $User_a$ is located.

The procedure to acquire the necessary information to reach the mobile user is commonly known as location tracking and is often supplemented by a location registration and deregistration procedure. These procedures will be studied in more details in the next paragraph.

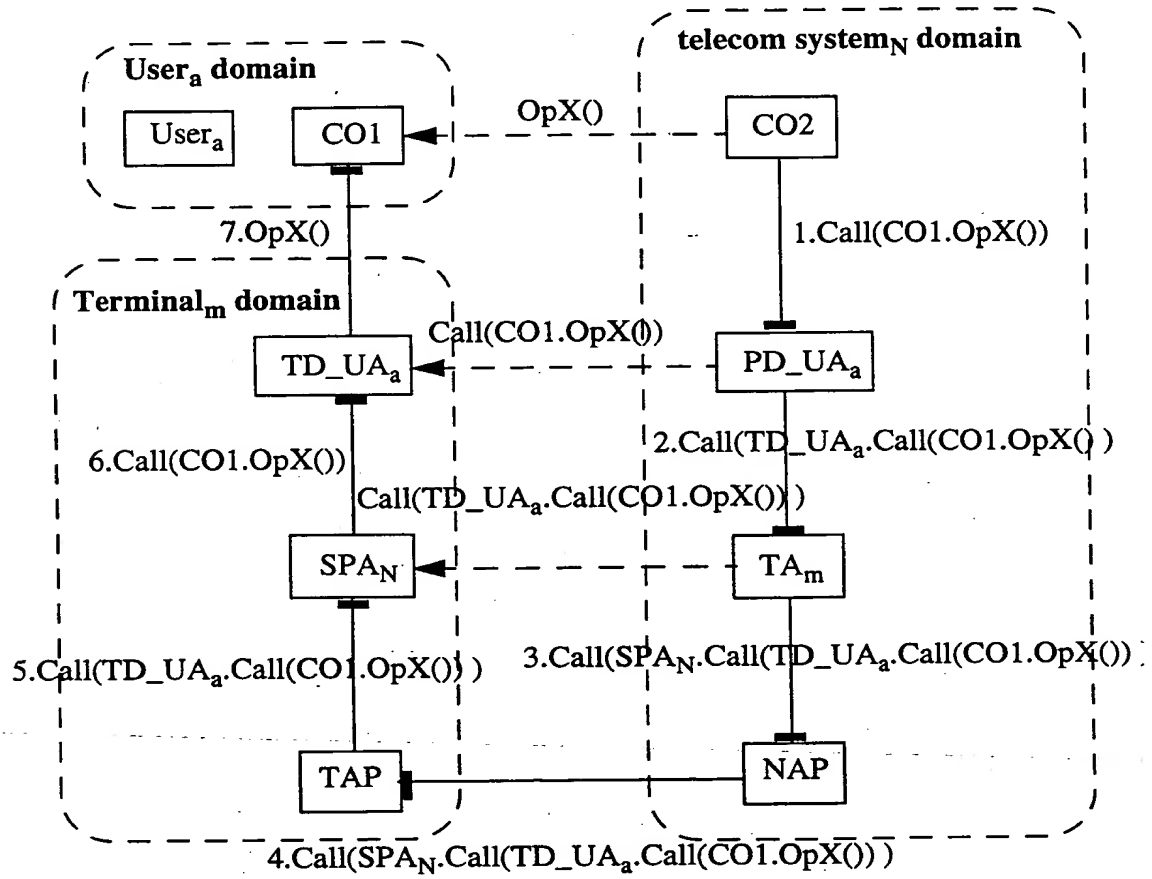


Figure 6.5 Operations initiated by an object in the telecom system domain

Let us suppose for the time being that the PD_UA_a somehow gets sufficient information and calls the correct Terminal_agent, namely TA_m. From this step to step 6, all the operations are related to the handling of terminal mobility. The TA_m will transfer the call to its peer-to-peer object SPA_N by issuing the operation `Call(`SPA_N`.Call(TD_UA`$_a$`.Call(CO1.OpX()))` to the NAP. The NAP transfers the call to the corresponding TAP. The TAP invokes then the operation

`Call(TD_UA`$_a$`.Call(CO1.OpX()))` of the SPA_N. The SPA_N calls the operation `Call(CO1.OpX())` of the TD_UA_a. Finally, the TD_UA_a invokes the operation `OpX()` of the object `CO1`. This concludes the convey of an operation initiated by an object in the telecom system domain.

We have seen that the condition for the success of these procedures is the availability of information about the current terminal which the user is using or having access to. Formally speaking, the success of operations initiated by objects residing on the telecom system domain depends on the definition of the association between the PD_UA_a and a TA.

When the $User_a$ is moving, the association between PD_UA_a and TA is changing correlatively and may sometimes be undefined. The operations necessary to determine this association are parts of the procedures commonly referred to as user registration and deregistration. There is here an analogy with the terminal registration and deregistration presented in Paragraph 5.2.4 and some of the discussions can also be applied here. This justifies the grouping of all mobility functions into one system, namely the Generic Mobility System (GMS) as proposed in this thesis.

Before continuing, it is important to emphasize that in this chapter, we focus only on the necessary data and functions which the GMS must have in order to route, convey and deliver the requested operations correctly to the addressed objects. These form only the basic part of the entire registration mechanism which is necessary to deliver applications to the user. Indeed, in order to be able to deliver applications to the user, communications at the object level with the user domain must be enabled. However, it is not said that once communications at object level are established, applications can be delivered to the user. The user may just want to access the telecom system domain in order to do some jobs but does not want to be disturbed or receive anything. Furthermore, there are multiple methods to do the registration for the application such as global registration for all applications, selective registration for each application, time table registration, alternative registration, etc.

To distinguish these two types of registrations, we use the terms ***User location Registration*** or just ***User Registration*** for "low level" registration and the terms ***Application Registration*** for the registration to enable the delivery of applications.

The Application Registration will be studied in Chapter 9 when the application as an entire set of computational objects is studied. The user location registration and deregistration will be studied in the next paragraph.

6.4.3 User location registration and deregistration

The user location registration and deregistration is the procedure to determine the association between the PD_UA and the TA.

6.4.3.1 The "on-the-Fly" method

As mentioned in Paragraph 5.2.4, the most straightforward method is the "on-the-fly" method. This method can also be called "lazy" since nothing is to be done unless it is

necessary, e.g a request is issued. When an object CO2 residing in the telecom system domain wants to call an operation of an object CO1 belonging to the $User_a$ domain, a request is issued to the PD_UA_a object. The PD_UA_a object will issue a broadcast to all the TAs asking them to search for the mobile user. The TAs can ask their counterpart, namely the SPA, to search for the corresponding TD_UA_a. If none of the SPAs succeed, then the $User_a$ cannot be reached. Nothing else can be done. If one SPA succeeds in finding the mobile $User_a$, it will answer to the corresponding TA which again informs the PD_UA_a. Then the PD_UA_a can send the operation invocation to the TA in question which forwards it to the corresponding SPA. The SPA forwards the operation to the TD_UA_a object which delivers it to the object CO1. Interactions are then enabled.

As mentioned in Chapter 5, this method is acceptable for small and "less geographically distributed" system, i.e system with small number of terminals and small number of users. It can be time consuming for larger or more geographically distributed systems. It may take a while to find the user or to know that it is not possible to find him. This method requires also much activity in the telecom system domain. If there are n terminals and m users in the system, then, in the worst case, $m \times n$ search procedures can be initiated simultaneously to find all the m users. This method is used in Internet when a search engine is used to identify the location of another web page or place of information.

6.4.3.2 Method based on the predetermination of the association between the PD_UA and the TA.

There is another method based on the predetermination of the association between the PD_UA and the TA. The PD_UA knows in advance whether the mobile user has contact with any TA or not, and which specific TA instance it is associated with. In other words, the PD_UA object has the ability to store the association between the PD_UA and the TA.

A simple implementation is to use a TA identifier which is a kind of "distribution transparent" pointer to a TA. By this method, much time will be saved at run-time when an operation request is issued.

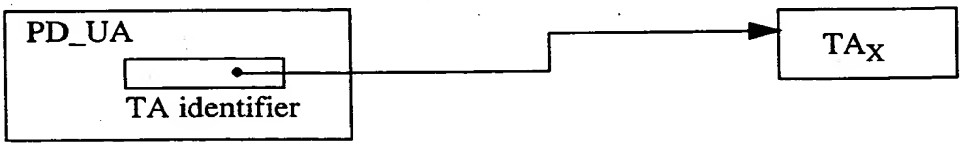


Figure 6.6 The PD_UA having a TA identifier

The TA identifier does not necessarily have to be incorporated inside the PD_UA but can be realized as a separate computational object, say User_Registration offering registration information service to the PD_UA object. The

User_Registration object may contain several TA identifiers since a user may be allowed to use several terminals at the same time.

It is worth recalling that the registration discussed here is a registration at object communication level which is different from the application registration. Application registration is used in the delivery of incoming applications to the user and will be treated in Chapter 9. Here, the registration of the user means that he is using a terminal and there is at least one TA associated with his PD_UA. When all the TA identifiers are Nil, it means that the user is not accessing the telecom system domain. Note that the user may use a terminal and have a TA associated with his PD_UA but still be not registered for any incoming application as is the case where he does not want to be disturbed.

The second piece of information required is the identifier of the corresponding TD_UA. Although this information can be deduced from the identity of the user and the identity of the terminal, it is more convenient to store it. The less the processing required at run-time, the shorter response time will be obtained. A User_Registration object containing a information about the TD_UAs and the TAs is shown in Figure 6.7.

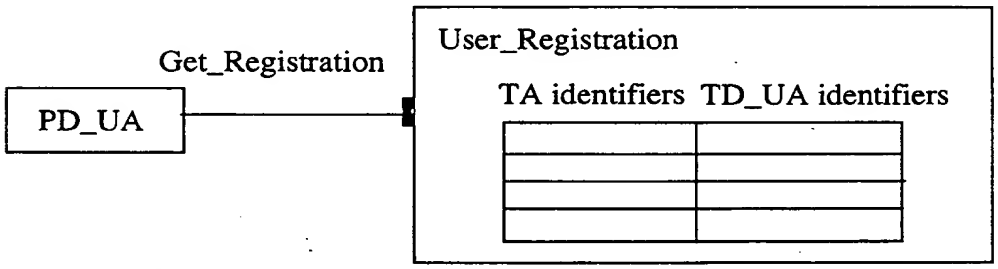


Figure 6.7 The registration data is stored in a dedicated object User_Registration

The questions now are: how can the PD_UA get the information about the TD_UAs and the TAs and when should the updating be done to ensure consistency with the real movements of the user?

6.4.3.3 Default registration

As assumed earlier in this chapter, every terminal is associated with a default user who is the owner of the terminal. In this case, the association between the PD_UA and the TA is always defined. We can refer to this as default registration.

Let us consider again the example where an object CO2 wants to invoke the operation OpX of an object CO1 belonging the $User_a$ domain. Let us suppose now that the

$User_a$ is the default user of the $Terminal_m$. When receiving the operation request `Call(CO1.OpX())`, the **PD_UA$_a$** call the operation `Get_Registration()` of the **User_Registration$_a$**. The **User_Registration$_a$** checks its **TD_UA** and **TA** table and return the identifiers of the default **TD_UA** and the **TA**, namely **TD_UA$_a$** and **TA$_m$** to **PD_UA$_a$**. The **PD_UA$_a$** proceeds to call the operation `Call(TD_UA`$_a$`.Call(CO1.OpX()))` on the **TA$_m$** object. The procedure will continue as described earlier until the whole operation `OpX` is accomplished.

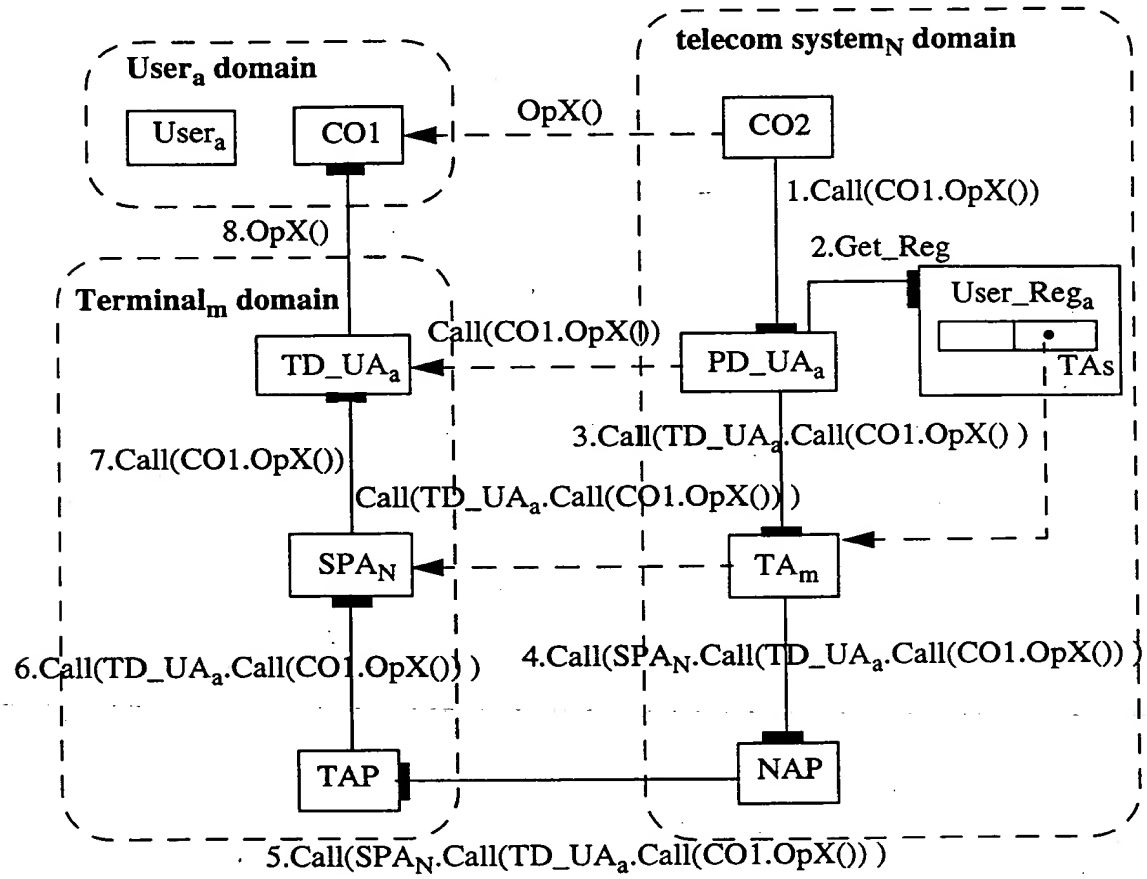


Figure 6.8 Interactions with the default user domain

6.4.3.4 Local registration

The local registration is the case where a $User_a$ wants to use a $Terminal_m$ belonging to a $User_m$. Actually the local registration belongs to the Access Session which purpose is to enable the user to access services. The Access Session will be described thoroughly in Chapter 9. For the time being, we focus only on the operations which

are directly related to the user registration and necessary to enable interactions between objects in the user domain and the telecom system domain.

Before continuing, some requirements have to be imposed on the terminal. In order to allow several users to use the same terminal to access the telecom system domain, the terminal must have some more local "intelligence", i.e more processing and storage capability than the current terminal used in the fixed network. We introduce an object called `UI` (User_Interface) in the $Terminal_m$ domain. This object is responsible for the supervision and control of interactions between the terminal and the users. `UI` has the functionality to switch between different users in the case where several users share the same terminal. Upon request, `UI` can also create new `TD_UA` object to serve new arriving user.

In Figure 6.9 we show an example with three users. $User_b$ and $User_c$ are already registered at $Terminal_m$. Upon request, `UI` can switch between these two users, i.e connecting the input/output channel respectively to TD_UA_b or TD_UA_c. The switch request can be realized in different ways such as entering command `New_User()`, depressing a key or by using a mouse to click on an icon, etc.

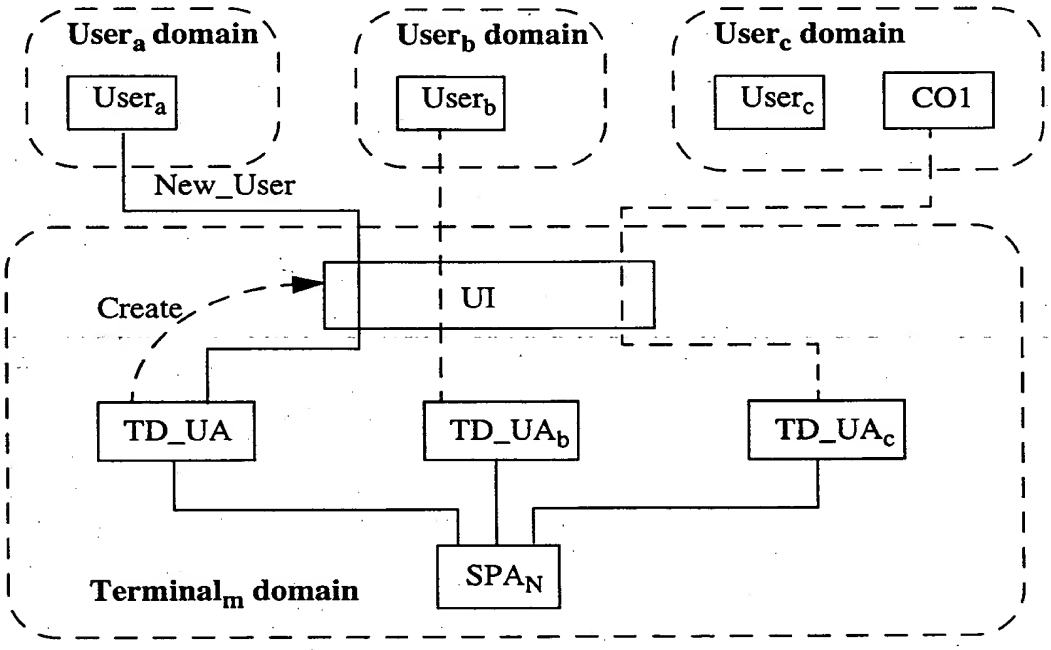


Figure 6.9 Functionality of the UI object

When $User_a$ arrives and wants to use the terminal he/she can issue a request New_User. The `UI` can create a new and "temporary" `TD_UA` object. By temporary it is meant that this `TD_UA` is not yet associated with any user or more specific to

any PD_UA which is known and recognisable for the telecom system domain. The association will only be done after the local registration has been accomplished successfully. The UI will also connect the input/output channel to this TD_UA. The $User_a$ can now interact with the temporary TD_UA.

Let us now return to our example. The $User_a$ wants to register himself at the $Terminal_m$. After calling the operation New_User() and getting a temporary TD_UA assigned to him, the $User_a$ enter his User Identifier (or name) to the temporary TD_UA.

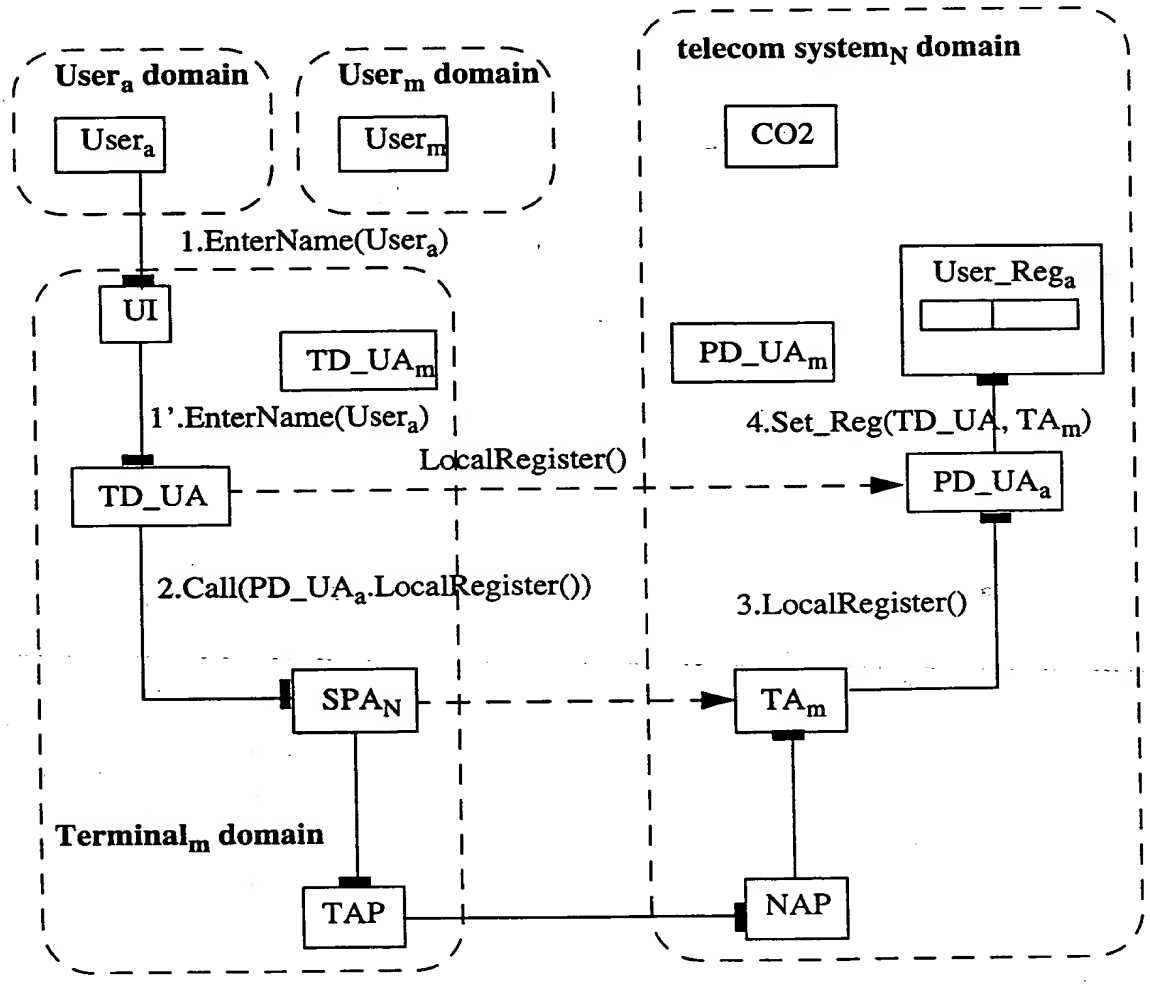


Figure 6.10 Local registration

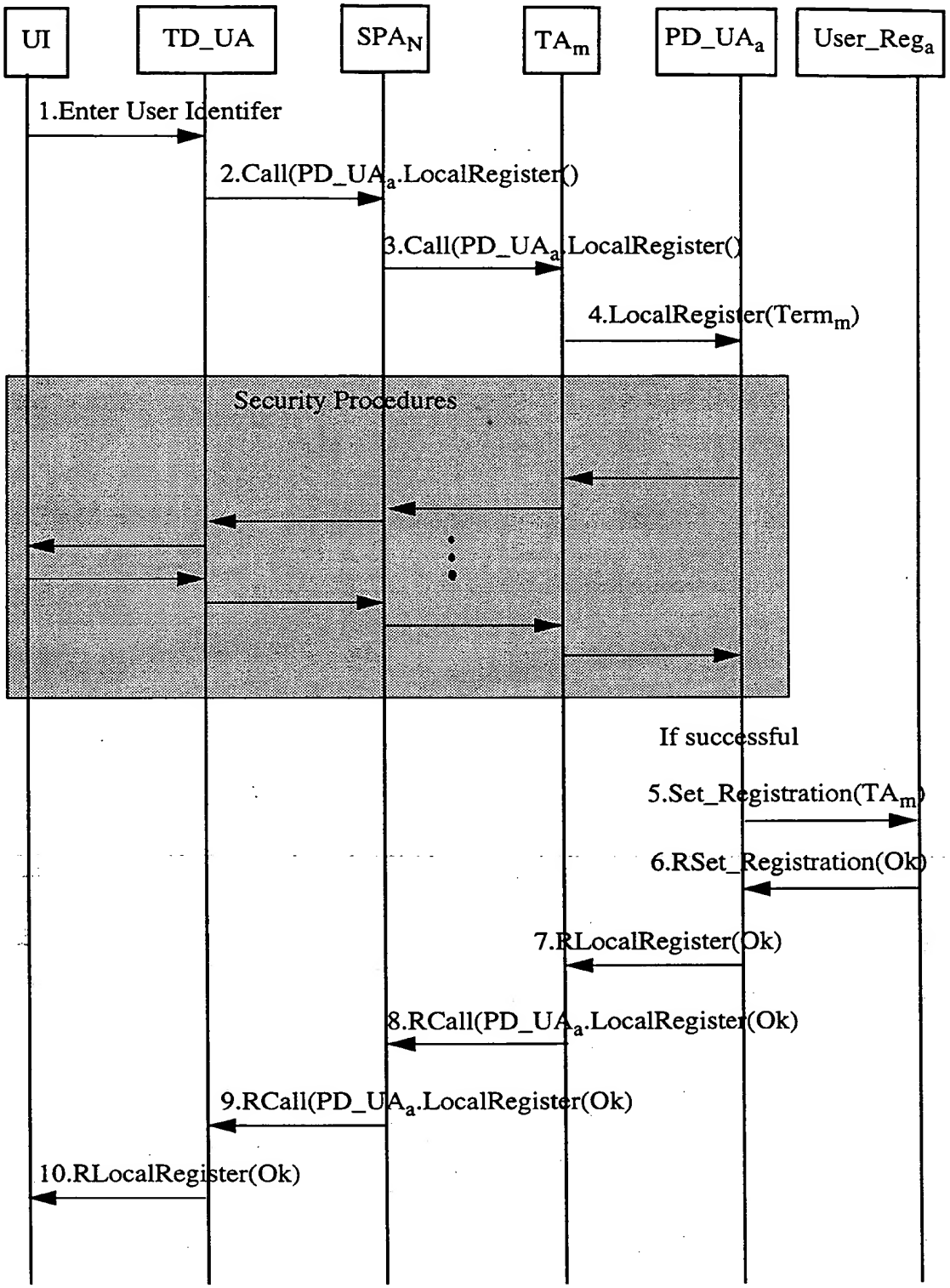


Figure 6.11 The procedure for the local registration

From the User Identifier the temporary TD_UA deduces the Computational Interface Identifier (CII) of the corresponding PD_UA_a knowing that the operation Local-Register() has to be invoked onthat PD_UA_a. The temporary TD_UA invokes therefore the operation Call(PD_UA$_a$.LocalRegister()) on the SPA_N, to start the convey of the operation LocalRegister(). The invocation is conveyed through the TAP, NAP and arrives at TA_m (see Chapter 5). The TA_m will then invoke the LocalRegister() operation on the PD_UA_a. The PD_UA_a will start the security procedures.

As shown in Figure 5.19, we assume now that the security procedures are successful. The PD_UA_a will call the operation Set_Registration(TD_UA, TA_m) on the $User\_Reg_a$ which saves the identifiers of the TD_UA and TA_m. The PD_UA_a will answer back to the TD_UA with a status Ok. The TD_UA will save the identifier of PD_UA_a and becomes a permanent agent associated to $User_a$. The TD_UA will remain alive until an eventual deregistration of the $User_a$ for $Terminal_m$.

6.4.3.5 Remote registration

The remote registration is the case where a $User_a$ wants to run or receive an application on a remote $Terminal_m$ where he has not logged in. $User_a$ is actually logged in at a $Terminal_n$. In order to perform a remote registration $User_a$ must have a registration application running on $Terminal_n$. A registration application is a special outgoing application which allows the registration for the delivery of application. We will come back to the registration application in Paragraph 9.5.2

Let us for the time being assume that the Registration application allows $User_a$ to enter the identity of the wanted remote terminal, namely $Terminal_m$.

The Registration application will then invoke Register($Term_m$) on the PD_UA_a.

The PD_UA_a will invoke the operation Register($User_a$) on TA_m, the terminal agent of $Terminal_m$. Here again, before granting the registration, the TA_m initiates the access control of the user for the use of the terminal (see Paragraph 7.4.3.1) to check whether $User_a$ is allowed to register himself at $Terminal_m$ for example to protect the privacy of the terminal owner. If the $User_a$ is not allowed, TA_m returns a NotAllowed status in the response to PD_UA_a which informs the registration application. The registration application will then deliver a NotAllowed message to the $User_a$.

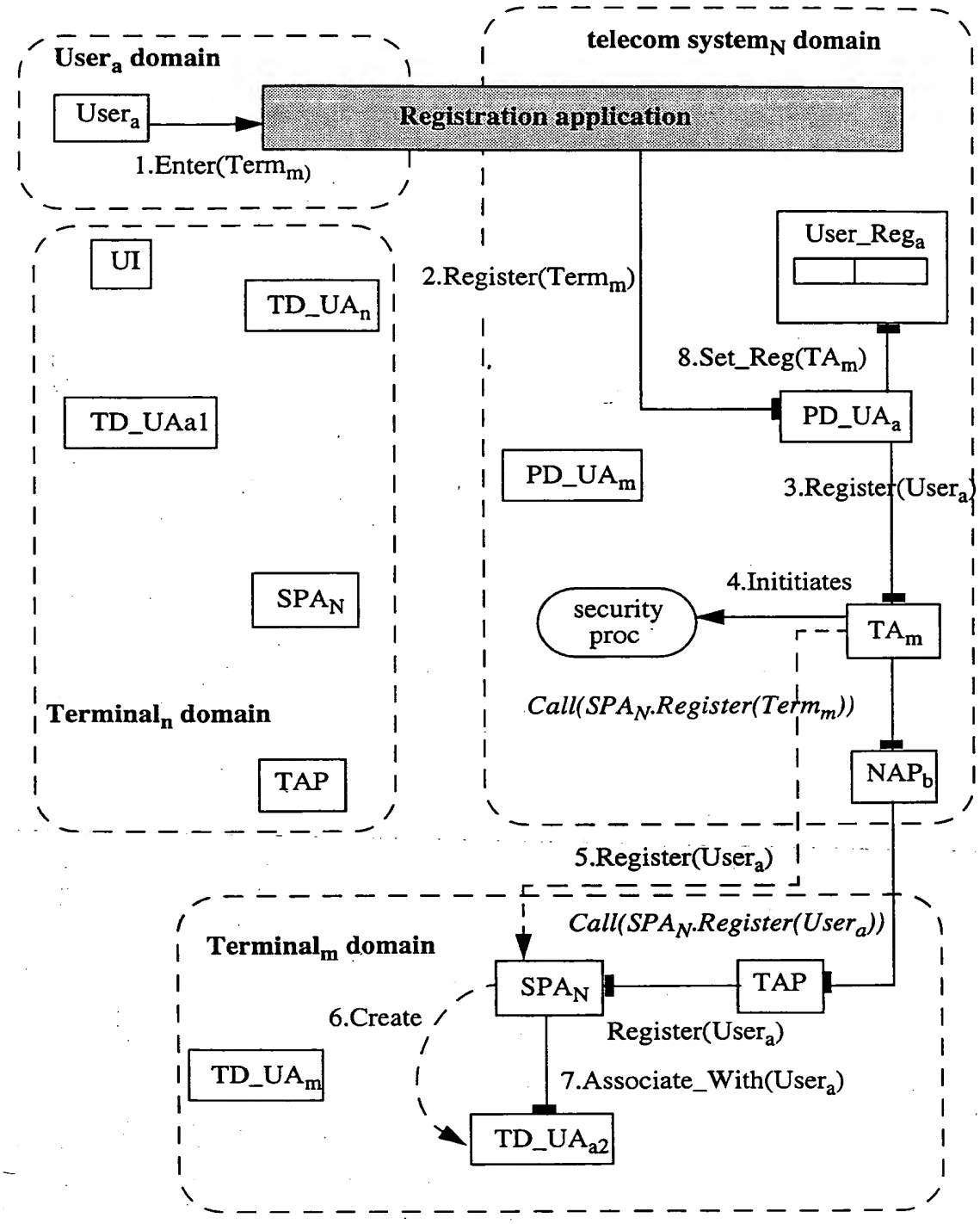


Figure 6.12 Remote registration

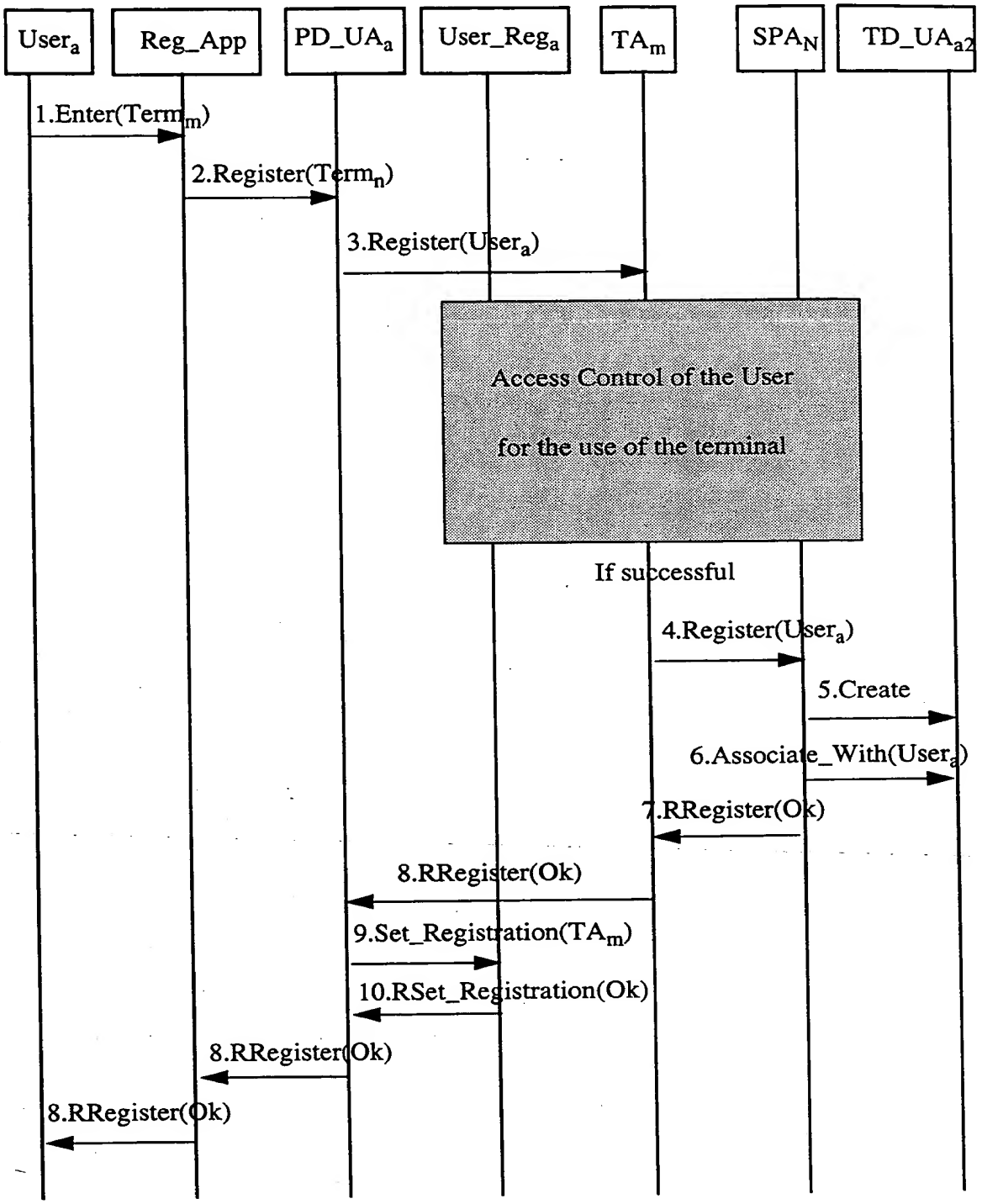


Figure 6.13 The procedure for the remote registration

Let us assume that the access control is successful. The TA_m will invoke the operation `Register(PD_UA_a)` on its counterpart SPA_N. As usual, the operation invocation must of course be conveyed through the NAP and TAP. In Figure 6.3, we present briefly some operations belonging to the terminal mobility support in italics in order not to burden our explanation.

Upon receipt of the call, the SPA_N will create a TD_UA_{a2} and invoke the operation `Associate_with(PD_UA_a)` on the TD_UA_{a2} causing the saving of the identifier of PD_UA_a. The TD_UA_{a2} will then persist until a deregistration for $Terminal_m$. The SPA_N will return an Ok status and the identifier of TD_UA_{a2} to the TA_m which also returns the same information to the PD_UA_a.

The PD_UA_a will then call the operation `Set_Registration(TD_UA_a2, TA_m)` on the $User\_Reg_a$ which saves the identifiers of TD_UA_{a2} and TA_m. The PD_UA_a returns then an Ok status to the Registration application. The Registration application delivers an Ok Status to the $User_a$ at $Terminal_m$. This completes the remote registration. The procedure for remote registration is shown in Figure 5.19.

6.4.3.6 Local deregistration

When the user wants to terminate all his activities at a terminal, he can just log out from this terminal. The deregistration is initiated locally from the same terminal.

Let us consider the example shown in Figure 6.3. A $User_a$ registered at the $Terminal_m$ wants to log out. He terminates his access session (see Paragraph 9.4) which results in a termination request to the TD_UA. Before terminating, the TD_UA invokes the operation `LocalDeregister()` on the PD_UA_a. The operation request is converted into the operation `Call(PD_UA_a.LocalRegister()` on the SPA_N. The operation is conveyed through the TAP, NAP and arrives at TA_m. The TA_m will then invoke the operation `LocalDeregister()` on the PD_UA_a. The PD_UA_a will invoke the operation `Reset_Registration(TD_UA, TA_m)` of the $User\_Reg_a$ which removes the identifiers of the TD_UA and TA_m from its internal table. The PD_UA_a will return to TD_UA a status Ok. The TD_UA can now terminate itself. The procedure for local deregistration is shown in Figure 5.19.

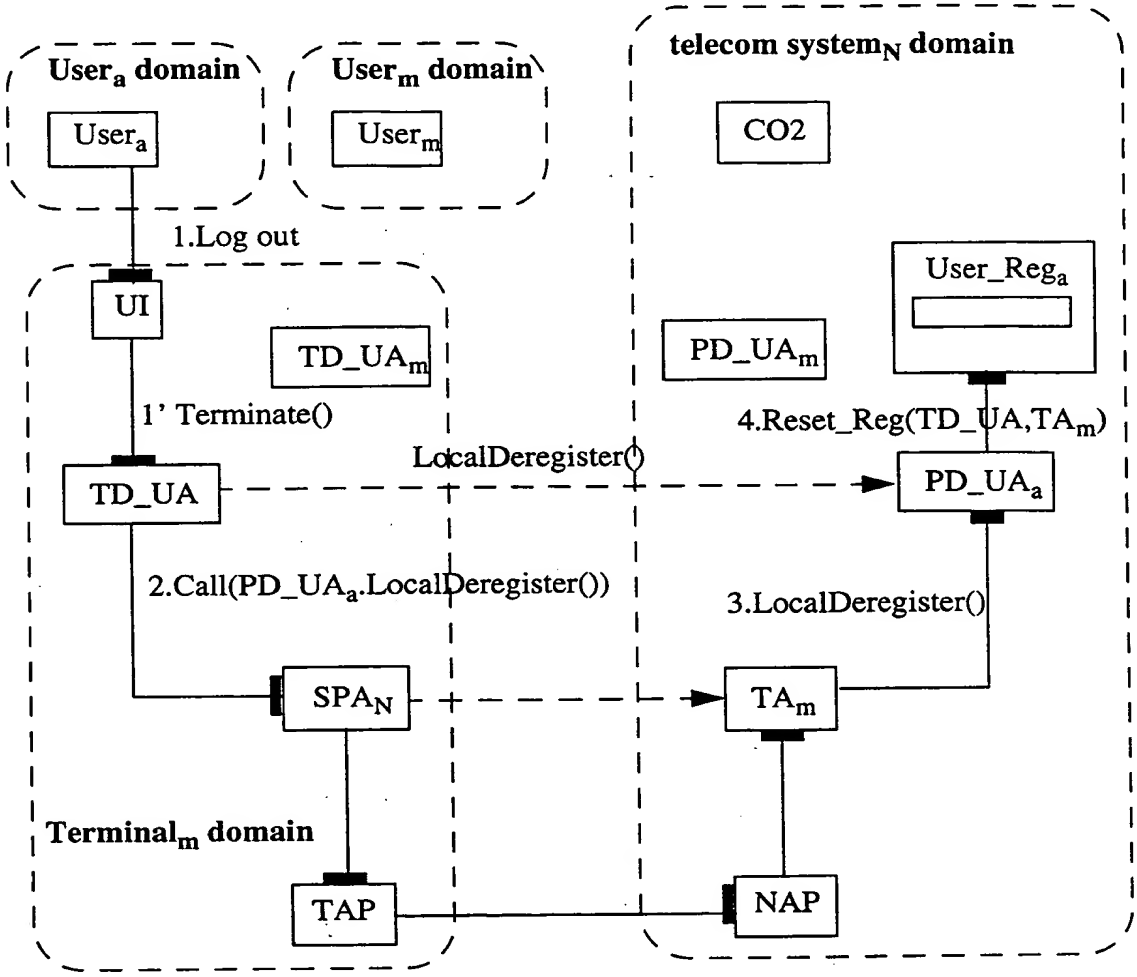


Figure 6.14 Local deregistration

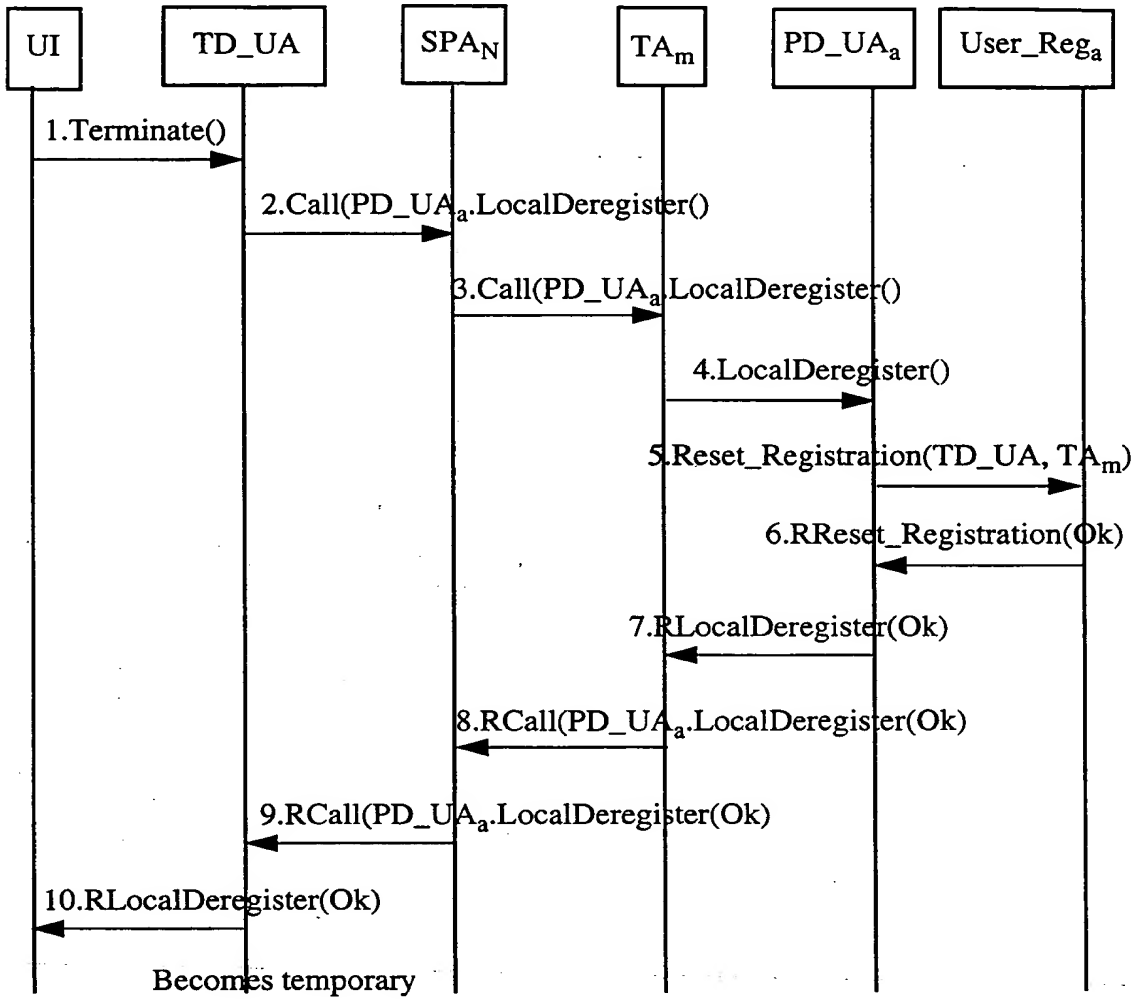


Figure 6.15 The procedure for the local registration

6.4.3.7 Remote deregistration

To terminate all his activity at a terminal, the user can also initiate the deregistration remotely from another terminal. Let us consider the example where a User$_a$ wants to deregister himself at a Terminal$_m$ belonging to a User$_m$ (the default user), from another terminalTerminal$_n$.

In order to do such deregistration User$_a$ must have access to a deregistration application running on Terminal$_n$ so that he can enter the identity of the remote terminal he wants to deregister from. The deregistration application will then invoke the operation `Deregister(Term`$_m$`)` on the **PD_UA$_a$** object.

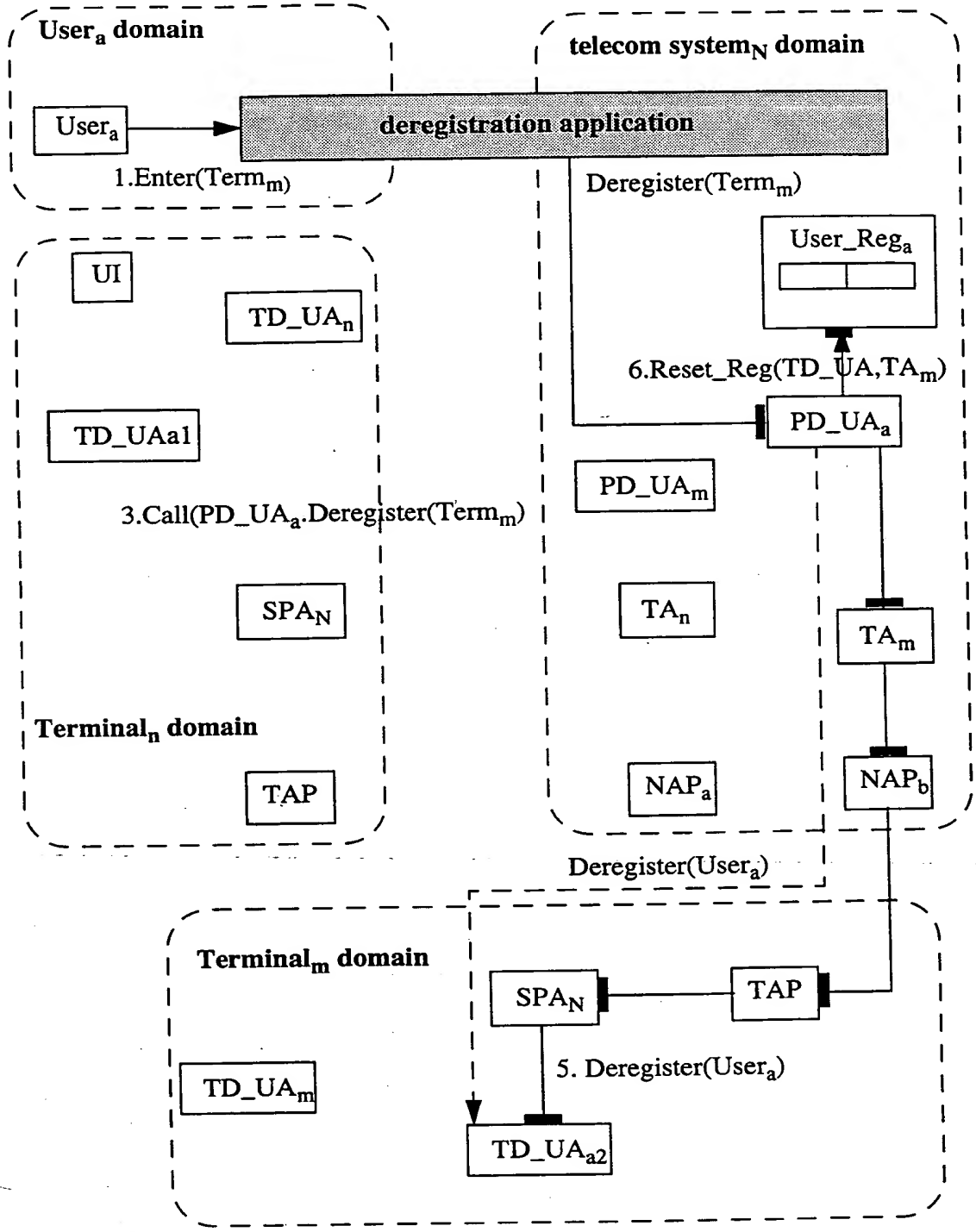


Figure 6.16 Remote deregistration

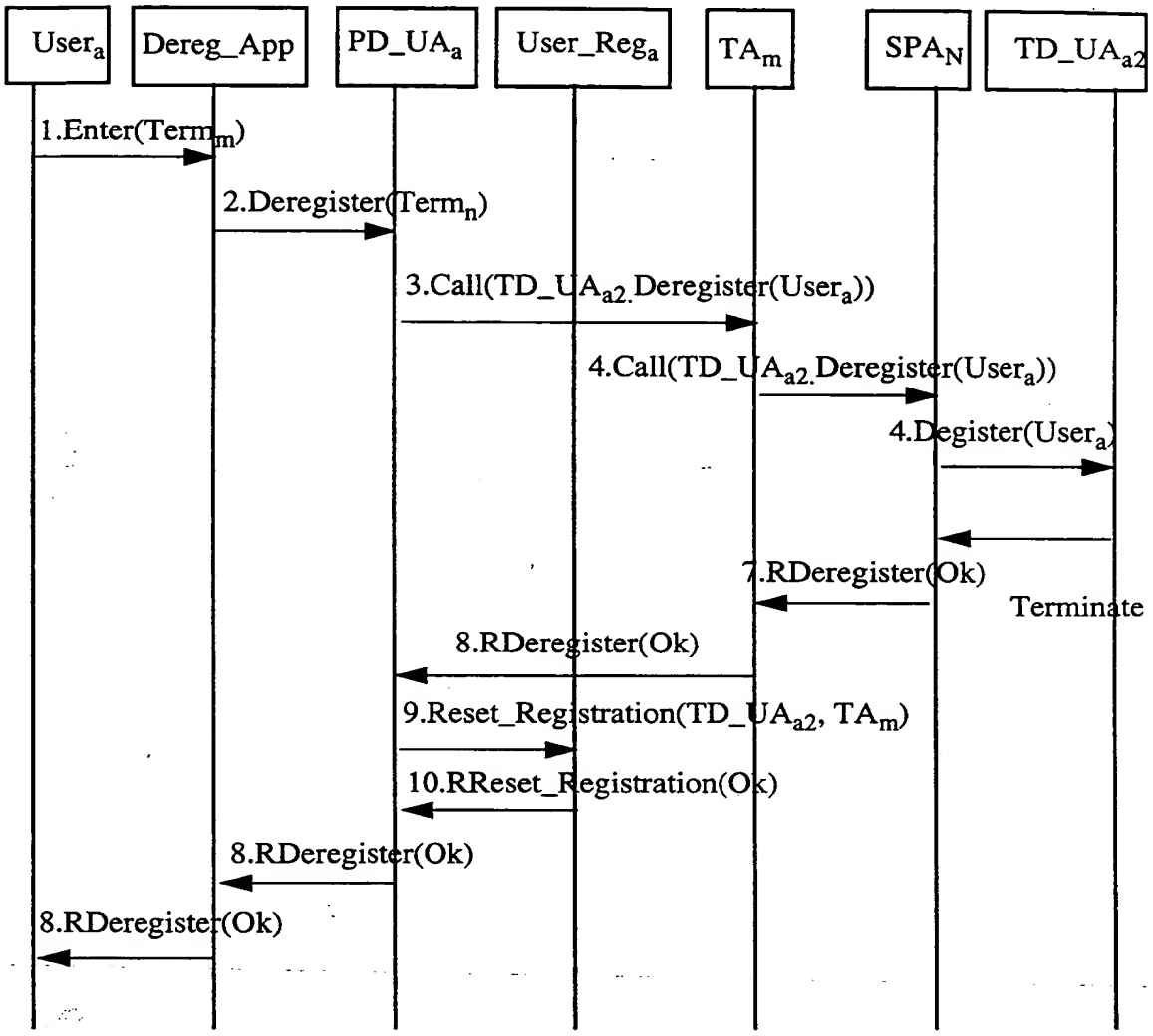


Figure 6.17 The procedure for the remote deregistration

The **PD_UA$_a$** will call the operation `Deregister(User`$_a$`)` on the **TD_UA$_{a2}$**, the terminal domain user agent on Terminal$_m$ assigned to User$_a$. This operation is shown by a dotted arrow in Figure 6.3. Physically it is converted to an operation on the **TA$_m$** and then conveyed via the **NAP**, **TAP**, **SPA$_N$** before arriving at **TD_UA$_{a2}$**. The **TD_UA$_{a2}$** object will remove the identifier of the **PD_UA$_a$**, return an `Ok` status to the **PD_UA$_a$** object and terminate itself.

The **PD_UA$_a$** object will call the operation `Reset_Registration(TD_UA`$_{a2}$`, TA`$_m$`)` on the **User_Reg$_a$** object causing the removal of the identifiers of the **TD_UA$_{a2}$** and **TA$_m$** from its internal table. Upon receipt of an `Ok` status, the **PD_UA$_a$** object can return the `Ok` status to the deregistration application which may forward it

to $User_a$. This completes the remote deregistration. The procedure for deregistration is shown in Figure 5.19

6.4.4 Supporting multiple terminal registration

As stated earlier, a user must be allowed to register at and use several terminals at the same time.

To clarify the situation, let us consider the example shown in Figure 6.3. A $User_a$ is registered at two terminals $Terminal_n$ and $Terminal_m$ and is associated with two terminal agents TD_UA_{a1} and TD_UA_{a2} respectively. He has two computational objects CO1 and CO3 running on $Terminal_n$ and $Terminal_m$ respectively. CO1 is interacting with CO2, CO3 with CO4, where CO2 and CO4 belong to the telecom system domain.

Suppose first that CO1 requests an operation `OpX()` on CO2. The operation request is translated to the operation `Call(CO2.OpX())` on TD_UA_{a1}. The TD_UA_{a1} object will then invoke the equivalent operation `Call(CO2.OpX())` on its peer-object PD_UA_a in the telecom system domain which in turn delivers the request to CO2. This is done without difficulty since TD_UA_{a1} knows the identifier of PD_UA_a. Similarly, CO3 can invoke any operation on CO4 without any problem. The existing mechanism is sufficient to support invocation initiated form the user domain. The opposite that is, object CO2 invoking an operation on CO1 or object CO4 on object CO3 requires additional information and functionality.

Suppose next that CO2 requests an operation `OpY()` on CO1. Again, the operation request is translated to the operation `Call(CO1.OpY())` on PD_UA_a. The problem now is that PD_UA_a does not know on which terminal CO1 is located. It is unable to decide whether it should transfer the operation request to TD_UA_{a1} or to TD_UA_{a2}. The PD_UA_a holds only the information that $User_a$ is registered at both terminals $Terminal_n$ and $Terminal_m$ but it has no knowledge concerning the location of each object of $User_a$. It is therefore obvious that such information must be supplied to it.

Every object in the user domain wishing to receive request from object on the telecom system domain must therefore be registered by some object in the telecom system domain so that the PD_UA_a object can interrogate in order to obtain the location of the user object. A convenient place to store this information will be the $User\_Registration_a$ object..

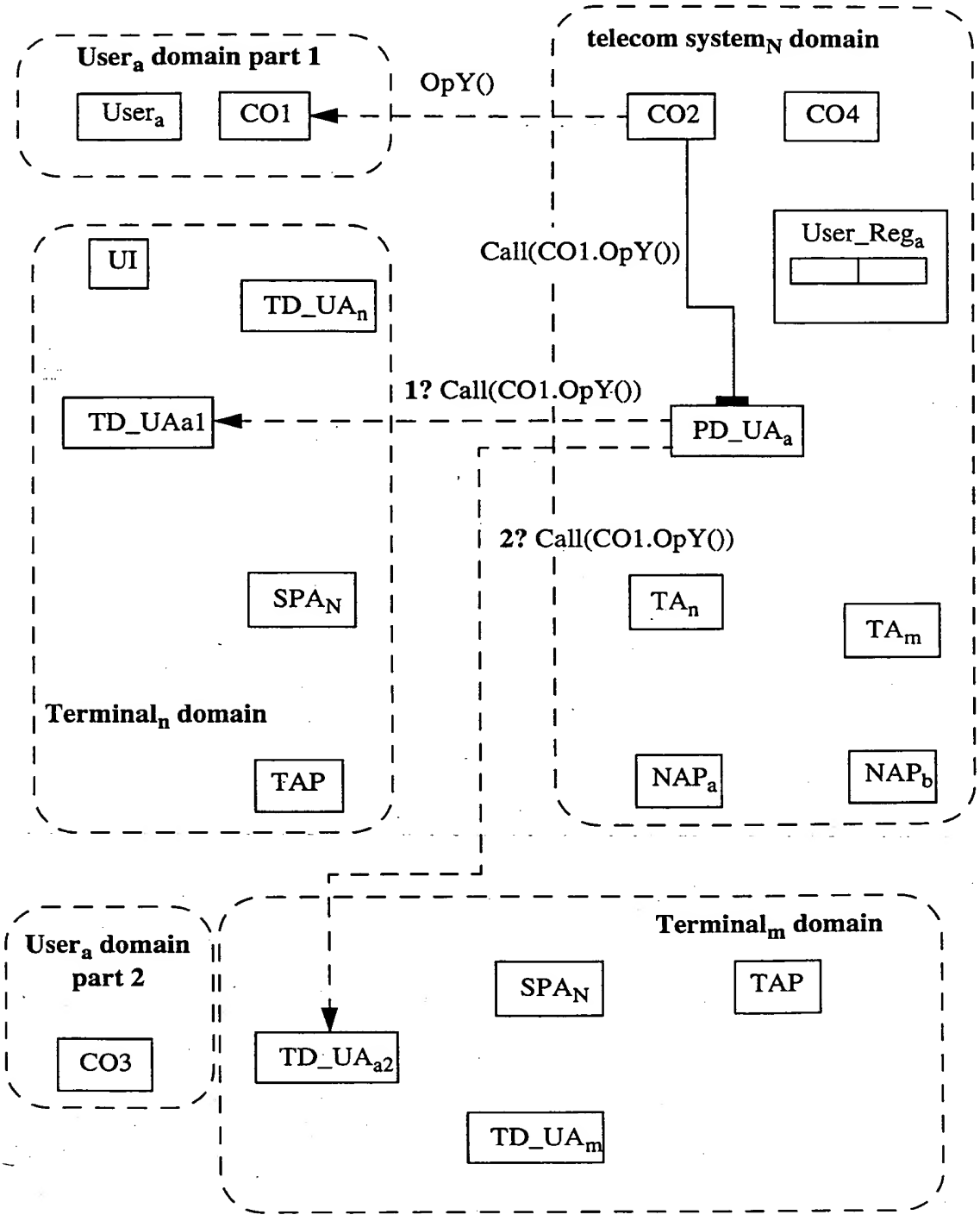


Figure 6.18 Example of multiple terminal registration

The user agents TD_UA_{a1} and TD_UA_{a2} must be equipped with an additional operation permitting the registration of the user objects in the terminal domain. Such operation, say **ObjReg** may have the following IDL/ODL specification[1]:

```
ObjReg(in ObjectId objectname, in ObjectId useragentname)
```

objectname is the Computational Interface Identifier of the registered object.

useragentname is the Computational Interface Identifier of the user agent to where the object will be registered. This argument is not necessary when the registration is done locally and may therefore be empty.

This registration of the user objects (CO1 or CO3) may be done by the user objects themselves or by some proxy objects (see Paragraph 8.5).

The registration procedure for the object CO1 is shown in Figure 6.19. The user agents (TD_UA and PD_UA) act as relays for passing the registration information to the **User_registration** object.

1. IDL: Interface Definition Language defined in CORBA. ODL: Object Definition Language is defined by TINA-C which is an extension of IDL allowing the definition of object [TIN95h].

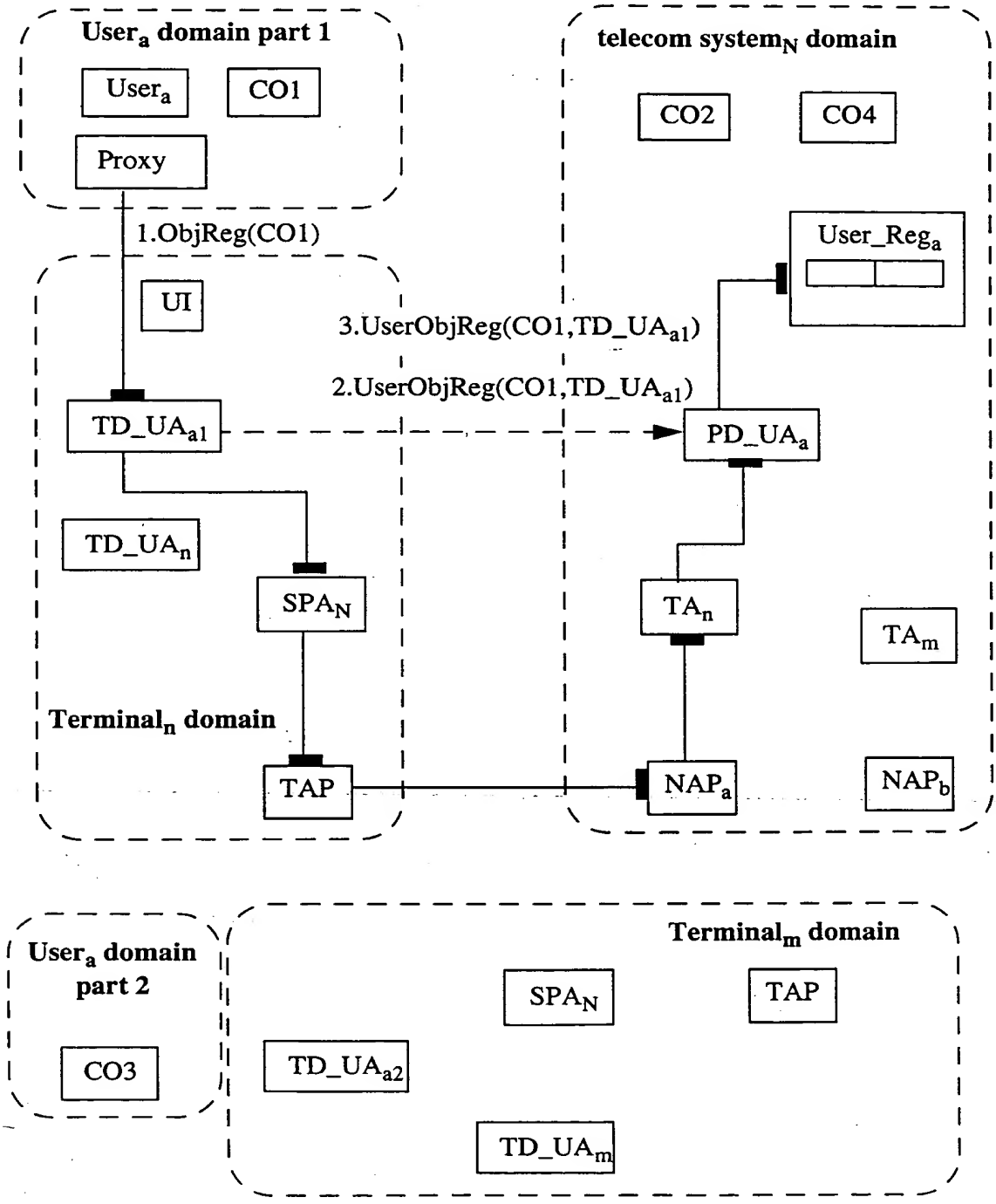


Figure 6.19 The registration of the user object CO1

When receiving the operation request `ObjReg(objectname,TD_UAname)`, the `TD_UA` will call the operation `UserObjReg(objectname, TD_UAname)` on its peer-object `PD_UA`. The `PD_UA` will in its turn call a similar operation `UserObjReg(objectname, TD_UAname)` on its `User_Registration` object. Upon receipt of the request, the `User_Registration` object shall save the object registration information. Its storage capability must therefore be expanded to include such information. The internal data structure can be implemented in different ways. An example is shown in Figure 6.20. For each row of the main table, there is assigned an `ObjectList` containing the identifiers of all object registered at a terminal.

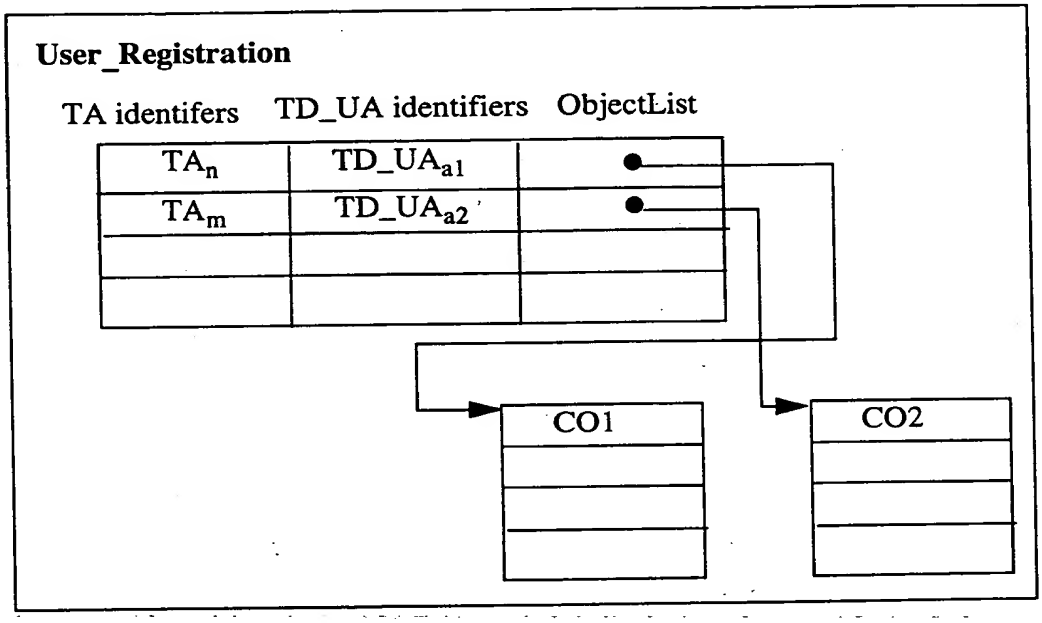


Figure 6.20 A User_registration object having object registration capability

To enhance further the functionality, a multiple object registration operation may also be defined for each of the objects `TD_UA`, `PD_UA` and `User_Registration`.

An object can also be deregistered. The procedure for object deregistration and the required operations are quite similar to these for registration. Upon receipt of an operation `UserObjDereg(objectname, TD_UAname)`, the `User_Registration` fetches the corresponding `ObjectList` and removes the identifier of the specified object. When a row of the main table is remove, i.e a user deregistration for a terminal is executed, the corresponding `ObjectList` will also be deleted.

In addition to the object registration and deregistration operations, the `User_registration` object must also have an operation ohter objects may use in order to request object registration information:

`GetObjReg(in ObjectId objectname, out ObjectId useragent-name)`

When receiving an operation request `Call(CO1.OpY())` from `CO2` addressed to `CO1`, and `CO1` is not in the telecom system domain, the PD_UA_a invokes `GetObjReg(CO1, AgentToCall)` on its `User_Registration` object. The identifier of TD_UA_{a1} is returned in the parameter `AgentToCall`. The PD_UA_a can proceed to invoke `Call(CO1.OpY())` on TD_UA_{a1}. The operation is finally conveyed all the way to TD_UA_{a1} which call `OpY` on `CO1`. The result will be returned using the same path back to `CO2`.

With all the described computational objects and functions, an arbitrary object belonging to the user domain can have operational interactions with any object on the telecom system domain. Our study concerning the support of operational interactions between the user domain and the telecom system domain is hence completed. We shall now examine the support of stream.

6.5 ENABLING STREAM BETWEEN THE USER DOMAIN AND THE TELECOM SYSTEM DOMAIN

It is not difficult to observe that once operational interactions are enabled between the user domain and the telecom system domain, streams are easily enabled, assuming that streams are already supported between the terminal domain and the telecom system domain.

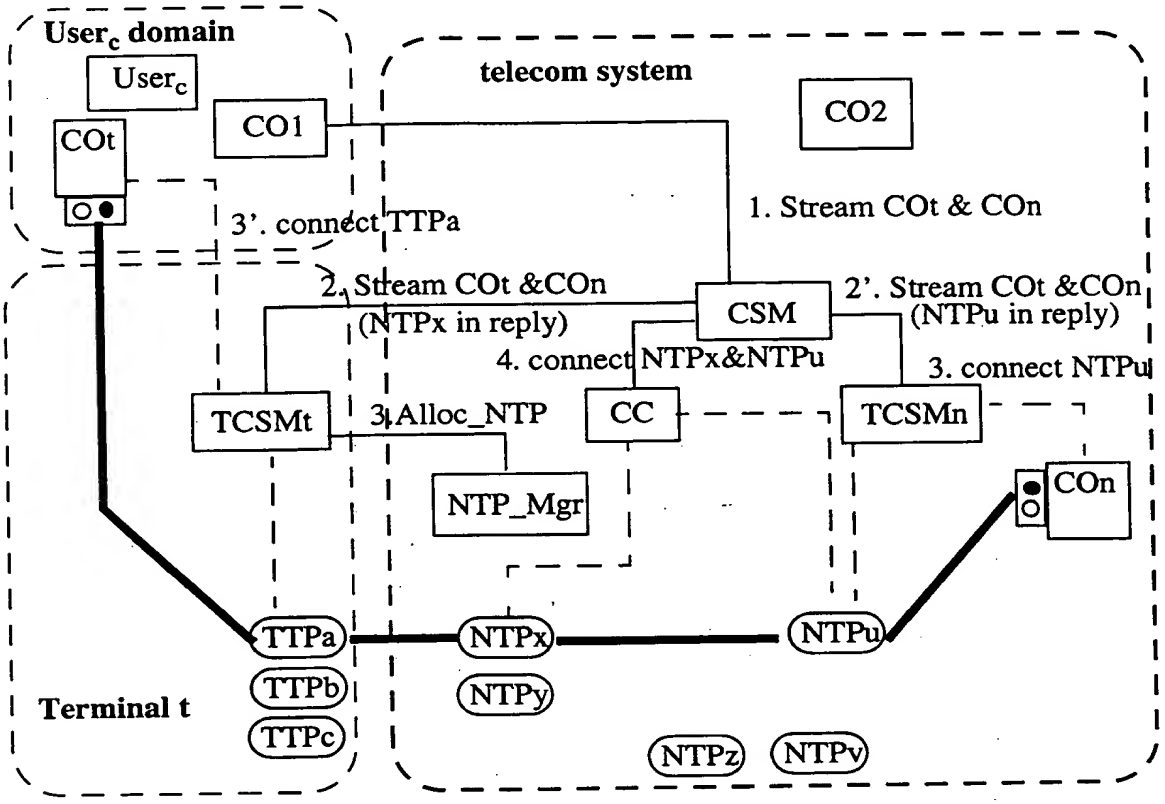


Figure 6.21 Stream establishment between the user domain and the telecom system domain

In fact, as shown in Figure 6.21, the conditions for the success of the stream establishment between object `COt` belonging to the user domain and object `COn` belonging to the telecom system are:

- First, the support of operational interactions between the user domain and the telecom system domain ensures that some object `CO1` of the user domain can invoke operation of the `CSM`, requesting a stream to be established.
- Second, the support of streams between the terminal domain and the telecom system domain is ensured by the mechanisms described in Paragraph 5.3.

Since these two conditions are satisfied, it is possible to conclude that streams between the user domain and the telecom system domain are supported without requiring additional functionality.

6.6 CONCLUSION

In order to support user mobility, the following computational objects are added to the GMS:

- In the user domain:
  - `TD_UA`
- In the terminal domain:
  - `UI` (User Interface)
- In the Telecom _System domain:
  - `PD_UA`
  - `User_Registration`

7

Security Issues

7.1 INTRODUCTION

Until now, the security aspects have not been considered and we have assumed that the communicating parties trust each other. As with any computing system, the telecom system is subject to multiple threats such as unauthorized use and disclosure, modification, destruction, etc. of information sent in the system [Pfl89]. Security violation causes loss to the users and may harm to the system. In fact, security plays a major role in the acceptance of a communication system by both the service provider and the user.

When the user is allowed to move and access to the telecommunication services anywhere and at any time, the risk of threats increases dramatically at the same time as the mechanisms necessary to enforce security become more difficult to realize. In systems supporting general mobility, fraudulent use of anyone's subscription can be attempted from any terminal and at any network access point. In this way the user may be exposed to various forms of fraud as, for example, fraudulent use of the user's resources by unauthorized parties who manage to take up the identity of the user, eavesdropping, unauthorized tapping or modification of information exchanged during communication, and disclosure of the user's physical location [MSG94], [ITU94]. Another security problem arises because the user is allowed to use any terminal and at any network access point. Such a temporary usage may conflict with the use of the terminal by the terminal owners, also referred to as third parties [ETSa]. In principle, third parties should not suffer in terms of loss of privacy or freedom of actions as a result of activities by the mobile user. In addition, the introduction of mobility should in no way harm the serving network or degrade the quality of the offered services.

To protect a system and objects within the system against all threats, several security services such as integrity, confidentiality, non-repudiation, etc. must be considered [ISO93a]. In this chapter, we will only consider the services which are directly related to mobility and study what impact they may have on the Generic Mobility System (GMS). In fact, the security services can be considered as belonging to a "Generic Security System" offering security services both to platforms and applications, i.e a mid-

dleware similar to GMS proposed in this thesis. This idea will be pursued in the UNIK TELEREP programme referred to in Paragraph 1.2.

7.2 MOBILITY RELATED SECURITY SERVICES

The security services which are directly influenced by mobility and demand special attention are:

- Identification
- Authentication
- Access Control

Concerning the order of the service execution, it is obvious that an identification should be done first. It is, however, not always evident whether the access control or authentication should be next. Users may be authenticated before being allowed to obtain access control information which will permit access to resources subject to an access control policy. The Authentication service may pass the results of authentication to the access control service, or in fact initiate access control. This is reasonable since access control will be meaningless if authenticity is not established. However, the situation may be reversed such as in the case where a user is black listed and should be rejected immediately by the access control service before any authentication is attempted. The relationship between the authentication service and the access control service can be specified by an access control policy. Anyway, this does not pose any problem for our study because the most important point is that the user is granted access if and only if both the access control service and the authentication service have been accomplished successfully.

7.2.1 Identification

The Identification is the procedure used by the user to identify himself to the telecom system. Normally, the identification is not carried out as a separated operation but is incorporated in other operations, e.g by a register operation from the mobile terminal or by initiating a call (See Paragraph 5.2.4).

In order to enable mobility of the user, the permanent association between a user and a network access point must be removed. The mobile user must also have a unique identity which is different from the one of the network access point used. A user may have one or several identities depending on the system, service or application. The identity may be permanent or temporarily allocated depending on time, location, context, etc. The telecom system domain may also have to identify itself to the user and needs also to have an identity.

In the Global System for Mobile communications (GSM) [ETS94a], [GSM94a], [ETS94b], [ETS94c] [MP92] the mobile subscriber is associated with two identities. The IMSI (International Mobile Subscriber Identity) is the permanent identity allocated at subscription. The TMSI (Temporary Mobile Subscriber Identity) is a local and

temporary identity having a meaning only in a given location area and for limited period of time. The TSMI, accompanied by a Location Area Identifier (LAI) is used to identify a mobile subscriber uniquely on the radio path. The purpose of the use of two identities is to avoid the possibility for an intruder to identify which subscriber is using a given resource on the radio path. This allows both a high level of confidentiality for user data and protection against the tracing of a user's location. The IMSI or any information allowing a listener to derive the IMSI is not normally used as addressing means on the radio path. The TMSI is used instead.

7.2.2 Authentication

The authentication service provides the assurance of the claimed identity of an entity [ISO93b]. The entity subject to the authentication is called **principal** while the entity doing the authentication is called **entity authentication**. In our cases, when the authentication is mutual, both the user and the telecom system domain will have both roles of principal and entity authentication.

Further delegation can be done. The **claimant** is the entity which represents the principal for the purposes of authentication. The **verifier** is the entity which represents the entity requiring an authenticated identity. This concept will be used later and the claimant and verifier objects will be introduced for authentication of the terminal and user.

In order to accomplish their mission, the claimant and the verifier may need assistance from one or several **Trusted Third Parties** (TTP) which can be an authentication server offering authentication services. There are several alternatives concerning the Trusted Third Party involvement and the detailed study falls beyond the scope of this thesis.

There are two threats to authentication. **Masquerade** refers to the pretence by an entity to be a different entity. To counter masquerade, authentication must be used in conjunction with some form of integrity service, which binds the authenticated identity to the activity. **Replay** refers to the repetition of authentication information exchanged, to produce an unauthorized effect. Authentication mechanism may be built to be resistant to replay.

An authentication method relies upon the following principles:

- something known, e.g a password
- something possessed, e.g a magnetic card or a smart card
- some immutable characteristics, e.g biometric identifiers
- accepting that a third entity (trusted third party) has established authentication
- context e.g address of principal

An authentication method used to authenticate a principal must also fit his characteristics such as:

- passive characteristics, e.g fingerprint, retinal characteristics
- information exchange and processing capability

- information storage capability
- unique fixed location

Authentication mechanisms may be classified according to their vulnerability:

- **Class 0 (Unprotected)**: This class is the weakest authentication, vulnerable to disclosure of authentication information and replay. It is symmetric because both sides share the same authentication information, e.g password, PIN code, etc.
- **Class 1 (Protected against disclosure)**: The authentication information, possibly combined with the distinguishing identifier, is transformed by a function before the transfer.
- **Class 2 (Protected against disclosure and replay on different verifiers)**: This class is similar to class 1 but include a characteristic unique to intended verifier as input to the transformation function. This method is used in most automatic teller machines
- **Class 3 (Protected against disclosure and replay on the same verifier)**: A unique number, e.g random number, time stamp, counter, cryptographic chaining is used in the transformation function.
- **Class 4 (Protected against disclosure and replay on the same verifier or different verifiers)**: A challenge mechanism is used to counter replay attacks. In response to an authentication request, the verifier issues a challenge to the claimant in form of a data item with a unique value. The claimant transforms the challenge information and the claim authentication information under some function, and returns the result of this transformation to the verifier.

Example: GSM [GSM94a] [MSG94] uses an authentication mechanism of class 4.

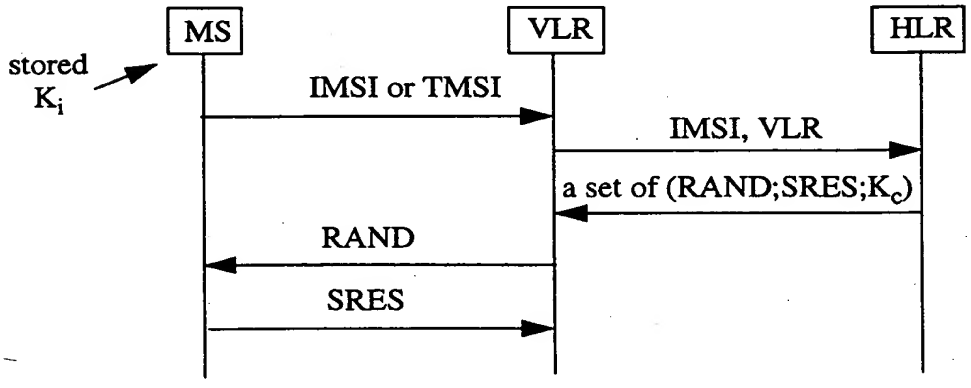


Figure 7.1 Authentication in GSM

When the MS (Mobile Station) updates its location, the VLR (Visitor Location

Register) contacts the MS's HLR and transmits the MS's IMSI to the HLR. The HLR asks its local Authentication Center for a set of triplets containing: a challenge (random number (RAND)), a signed response (SRES) and a corresponding session key (K_c). The parameters SRES and K_c are computed with the unpublished Algorithms A_3 and A_8 that implement a one-way function:

$$SRES = A_3(K_i, RAND)$$

$$K_c = A_8(K_i, RAND)$$

K_i is the secret key contained in the SIM card of the MS and known only by the authentication center associated with the HLR. The VLR sends the RAND to the MS and expects to receive an SRES. K_c is used to encipher data between the MS and the local Mobile Service Switching Center (MSC).

The choice of the proper authentication mechanism for each case falls beyond the scope of this thesis. Our goal is however to minimize the dependency of the GMS on the chosen authentication mechanism and to achieve flexibility.

7.2.3 Access Control

The primary goal of access control is to counter the threat of unauthorized operations involving a computer or communications system. These threats are frequently subdivided into classes known as unauthorized use, disclosure, modification, destruction and denial of service [ISO92] [Sko94].

The basic entities and functions involved in access control are the Initiator, the Access control Enforcement Function (AEF), the Access control Decision Function (ADF) and the target.

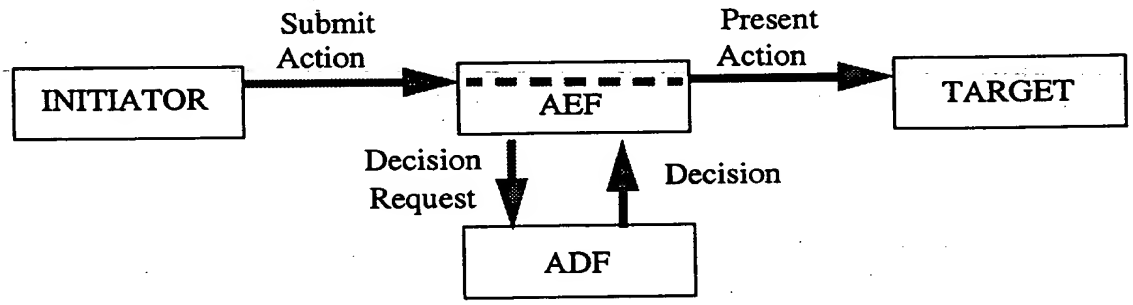


Figure 7.2 Illustration of fundamental access control

Initiators represent entities which are in our case the users and terminals that access or attempt to access targets. Targets represent entities that are accessed by initiators. In our case, a target may the telecom system domain, a terminal, an application or a telecommunication service.

The AEF ensures that only actions allowed, as determined by the ADF, are performed by the initiator on the target. When the initiator makes a request to perform a particular action on the target, the AEF informs the ADF that a decision is required so that a determination can be made.

In order to perform this decision, the ADF is provided with the action (as part of the decision request) and the following types of Access Control Decision Information (ADI):

- Initiator ADI (ADI derived from the ACI (Access Control Information) bound to the initiator)
- Target ADI (ADI derived from the ACI bound to the target)
- Action ADI (ADI derived from the ACI bound to the action)

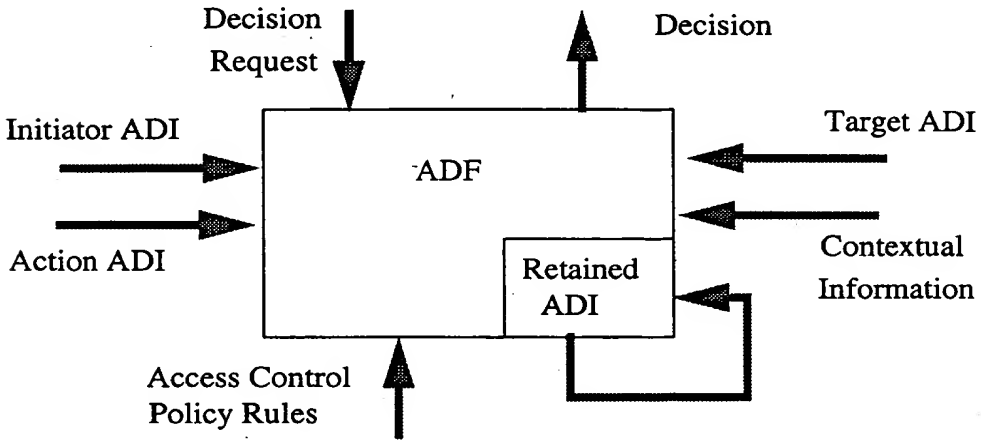


Figure 7.3 Illustration of ADF

The other inputs to the ADF are the Access Control Policy rules and any contextual information such as location of the initiator, time of access, etc.

Based on these inputs, and possibly from the ADI retained from prior decisions, the ADF arrives at a decision to allow or deny the initiator's attempted access to the target. The decision is conveyed to the AEF which then either allows the action to pass or takes other appropriate actions.

The access control in our context is the procedure used by the telecom system domain to ensure that the user accesses the telecom system domain in accordance with the restrictions specified at subscription. When mobility is supported, every user will have the possibility to use any terminals at any access points. The access control procedure is also intended to limit the access capability of a user for the protection and privacy of third party. The third party can be the owner of the terminal or the access

point, and must have the right to block or deblock, suspend or reset the service delivery at his terminal or access point to a user.

We shall in the following paragraphs study how the GMS objects cooperate with the security objects to perform identification, authentication and access control of the terminal and the user. To carry out the access control of the user, both the user profile and the terminal profile must be consulted. The GMS will assume this responsibility. For the authentication, the GMS may then choose the appropriate authentication method depending upon terminal capabilities, access point characteristics, etc. When only weak authentication (class 0 or class 1) is available, the GMS may also activate other protective mechanisms such as outgoing call screening (limiting the number and type of destinations that can be reached), credit limit (limiting the financial risk of the user) and "call jumping" detection.

We shall consider successively the security services related to terminal mobility and to user mobility.

7.3 SECURITY SERVICES RELATED TO TERMINAL MOBILITY

7.3.1 Identification

In order to allow terminal mobility, a terminal must have an identity which is different from both the Network Access Point (NAP) identity and the user identity. In addition, it must be possible to identify unambiguously a terminal from its Terminal Identity (TI). As already mentioned, a terminal may have several identities according to time, location, context, etc. A PC may, for example, be identified by an IP address, a telephone number and a data network number. Since mutual authentication is considered, the telecom system domain must also have an identity, TSI, recognisable by the terminal.

The identification of both terminal and telecom system domain is carried out at every attempt to (re)establish a connection. This connection can be a physical line or a radio path. A handshaking procedure is initiated between the TAP and the NAP. They exchange information such as terminal identity (TI) and service provider identity (SPI). After mutual identification the connection can be established so that information exchange or other security service can be invoked.

7.3.2 Authentication

7.3.2.1 Information model

As mentioned earlier, the authentication mechanism used to authenticate a terminal must fit its type and capability. There are several type of terminals demanding different authentication mechanisms. Some examples are:

- For a fixed terminal the location, which is identical to the NAP, may be suf-

ficient for authentication and the terminal does not need storage or exchange and processing capability. However, in order to avoid line tapping stronger authentication may be required.

- For a portable terminal which is allowed to be connected to a small set of sockets, a password may be sufficient for authentication because the risk of disclosure and replay is minimal due to the limited number of access points. The terminal must have limited storage capability. However, for the same reason as above, stronger authentication may be desired.
- For a wireless terminal the risk of disclosure and replay are very high because the radio path can be tapped and the operation range of the terminal is not confined to a restricted area. An authentication mechanism of class 4 should be used. This will require hence both storage, exchange and processing capability of the terminal.

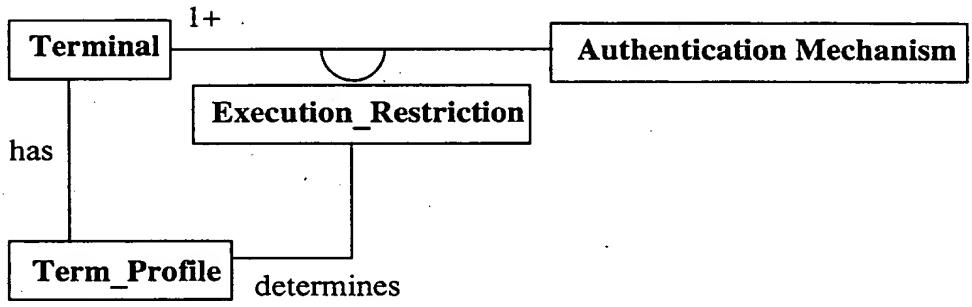


Figure 7.4 An information model for the terminal authentication

An information model for terminal authentication is shown in Figure 7.4. The relationship between the **Terminal** and the **Authentication_Mechanism** is called **Execution_Restriction** where the restrictions are determined by the **Term_Profile**. The **Term_Profile** object contains all the attributes which are specific to the terminal used. The **Term_profile** object can be decomposed into four component objects: **Capability**, **Usage_Restriction**, **Access_Rights** and **Security_Info** as shown Figure 7.5. The object **Security_Info** contains necessary information for security services (authentication, access control) such as authentication mechanism and secret key K_i. The object **Capability** contains attributes related to terminal capabilities such as memory, voice, graphic, etc. which may constrain the type of authentication methods that may be used towards the terminal. The object **Access_Rights** contains information used for the access control of the terminal. Such information can be, for example roaming restriction and time restriction. The **Usage_Restriction** contains information used for the access control for the use of the terminal and is intended for the protection of third party, and may contain information concerning conditional barring of all outgoing calls, barring of certain users, etc.

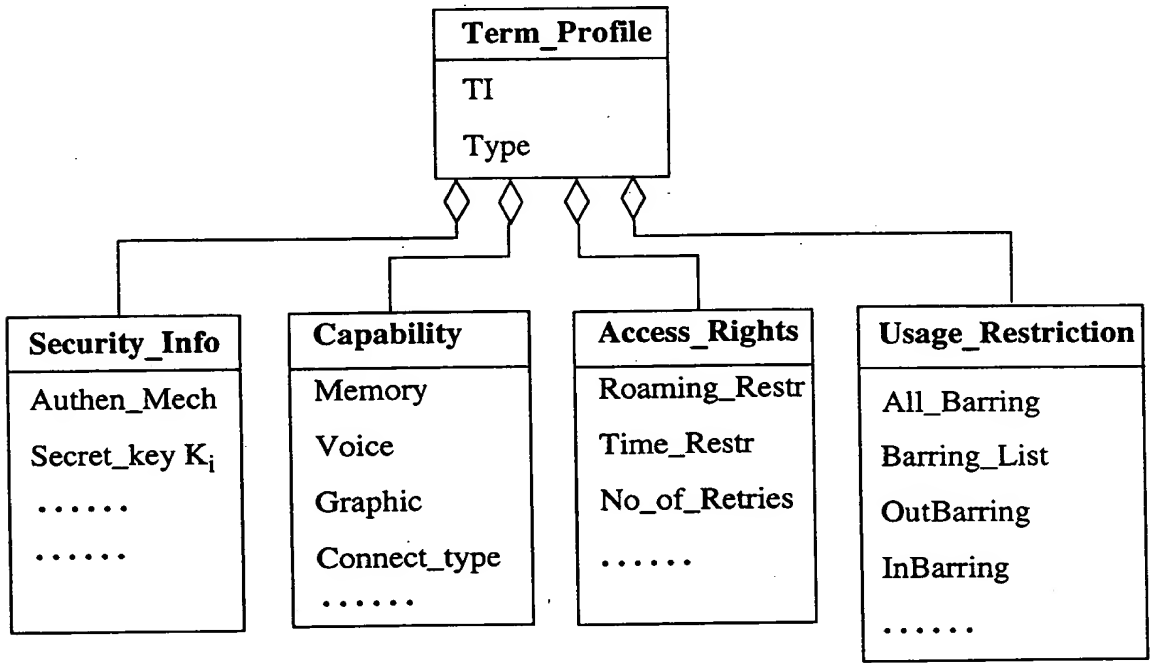


Figure 7.5 The Information object Term_Profile

7.3.2.2 Computational model

The information object **Term_Profile** is mapped into a computational object where we for simplicity is using the same name `Term_Profile`. As already mentioned, the authentication can be mutual and both the terminal and the telecom system domain can assume both roles of claimant and verifier. Four objects, `TD_Claimant`, `TD_Verifier`, `PD_Claimant` and `PD_Verifier` are therefore introduced to assure the authentication functions and exchanges. Note that in order to accomplish their mission, these security objects may need assistance from one or several Trusted Third Parties which may be some Authentication Server. However, the realization of the internal functions of these objects falls beyond the scope of the thesis. The most important point is that their interfaces with other GMS objects remain unchanged. To achieve flexibility, an common operation `Auth_Transfer` is defined for the four security objects `TD_Claimant`, `TD_Verifier`, etc. and also for the GMS objects `TA`, `NAP`, `TAP` and `SPA`. This operation has as first argument an authentication operation name and a list of related operational arguments. As will soon be apparent, the `Auth_Transfer` operation will form a chain of operations on successive objects. The returned result will follow the same chain but in opposite direction. On the `SPA` and `TA`, an operation `Auth_Result` is defined. This operation returns the result of the verification from the Verifier.

Since the authentication procedures are quite similar for both sides, we shall only study the authentication of the terminal. This case is shown in Figure 7.6.

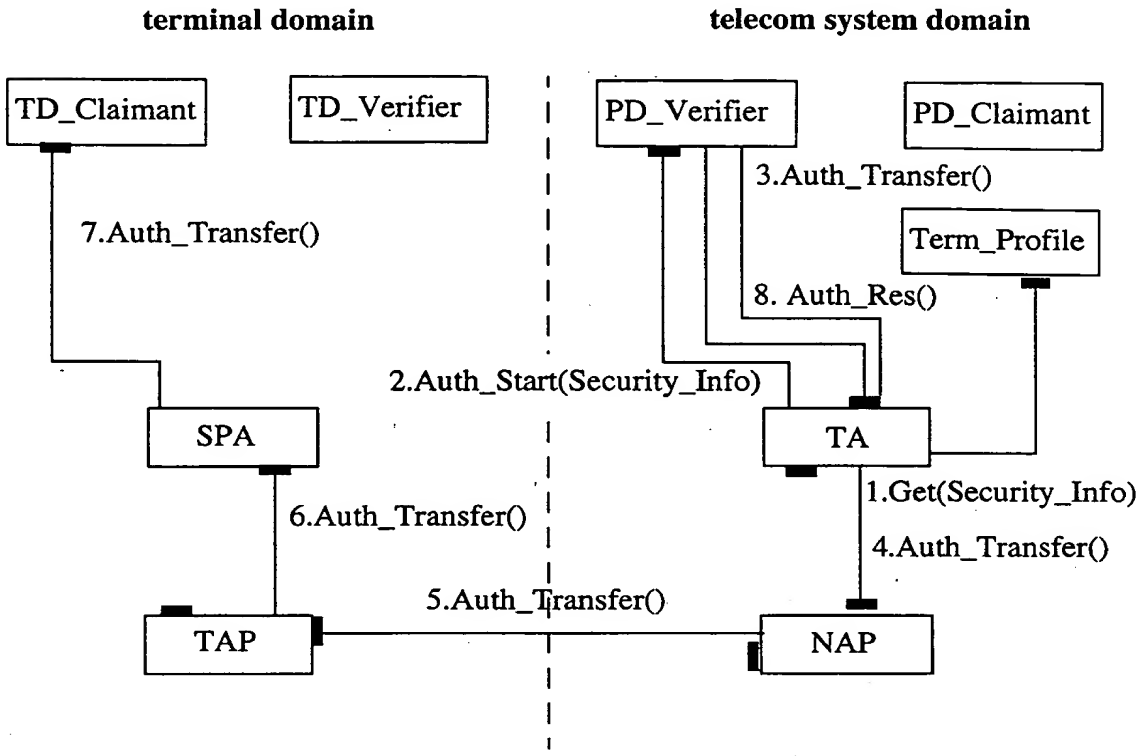


Figure 7.6 Computational model of the authentication of the terminal by the telecom system

As described in Chapter 5, the authentication of the terminal is initiated by the **TA** as a consequence of an operation request which depends on the states of the terminal. The authentication procedure is as follows:

1. **TA** invokes the operation `Get(Security_Info)` on **Term_Profile** to get necessary information to initiate the authentication

2. **TA** invokes the operation `Auth_Start(Security_info)` on **PD_Verifier** to start the authentication.

3: Depending on the `Security_Info` received, **PD_Verifier** can initiate different authentication mechanism. Therefore, different operations and arguments can be directed toward the principal, but are all incorporated in the same operation `Auth_Transfer()` invoked on the **TA**. This can be done since the arguments will only be read by the **TD_Claimant**.

4. TA invokes Auth_Transfer() on NAP.

5. NAP invokes Auth_Transfer() on TAP.

6. TAP invokes Auth_Transfer() on SPA.

7. SPA invokes Auth_transfer() on TD_Claimant.

This chain of operations follows the rules for exchanges of information between the telecom system domain and the terminal domain defined in Chapter 5. TD_Claimant generates the Claim Authentication Information (AI), i.e answers to the challenge issued by the verifier and invokes Auth_Transfer() on the SPA to send the result back. The Claim AI is conveyed through the TAP, NAP, TA and arrive at PD_Verifier. For the sake of clarity, all these operations are omitted in Figure 7.6. In fact, the PD_Verifier can issue several challenges to the TD_Claimant and several transfer rounds may be run before the verification can be started. This is sytem dependent and does not affect teh design of the GMS.

8. PD_Verifier does the verification and invokes Auth_Result on TA to indicate that the authentication was successful or unsuccessful. If authentication fails, the TA may terminate the interaction with the terminal. The decision to terminate or to continue may also left to the access control service. We propose the latter in order to avoid that security decisions to be programmed into the TA. In this case the TA will proceed with the access control service of the terminal. The authentication result and the authentication mechanism (or class) used will be transferred the access control mechanism as contextual information.

7.3.3 Access Control

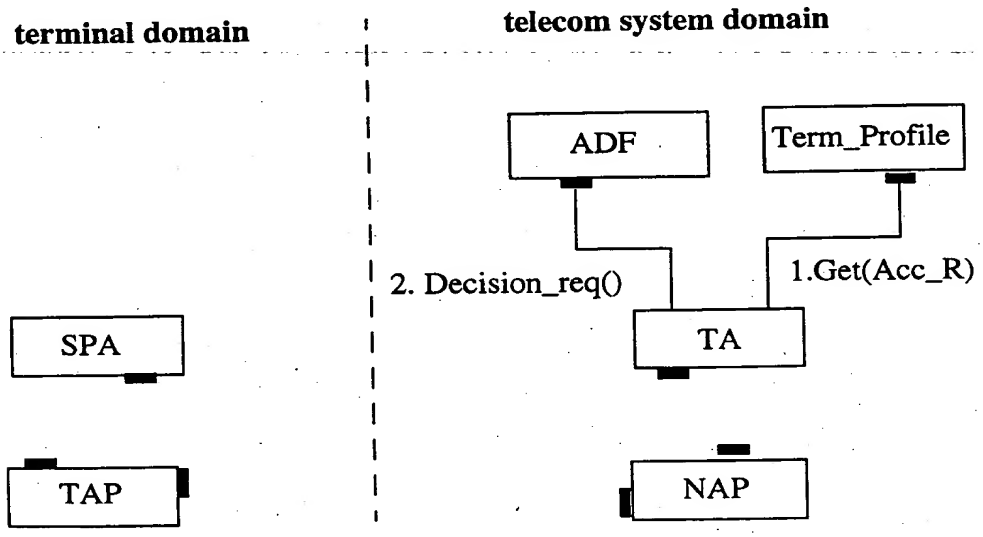


Figure 7.7 A computational model of the Access Control for the terminal

The purpose of the access control is to decide whether or not the terminal should be allowed to access the telecom system domain. The initiator is the terminal represented by the SPA. The target is the telecom system domain. The AEF is assumed by the TA. The ADF is required to be a separate object because this functionality cannot easily be included in any of the other objects in the model. The access of the terminal to the telecom system may be limited by some parameters such as Roaming_Restriction, Time_Restriction, etc. as shown in Figure 7.5. A computational model of the access control for the terminal is shown in Figure 7.7.

The access control procedure is as follows:

1. The TA invokes a Get(Access_Rights) on the Term_Profile object to acquire the access control Decision Information (ADI).

2. The TA invokes the operation Decision_Request on the ADF object. The arguments of this operation are the ADI obtained from Term_Profile object and the contextual information such as the identifier of the NAP used and the information concerning the preceding authentication procedure such as authentication successful/ unsuccessful and authentication mechanism (or class used).

The ADF may use the access control services offered by the platform or a security system to obtain further contextual information such as time, system status, etc. and the access control Policy Rules. This is not shown in the figure. The ADF makes the decision and returns the Access_Result to TA. The Access_result may be granted, not_granted or suspended. If the Access_result is granted, the TA may continue information exchanges with the terminal. If the Access_Result is Suspended, depending on the access control Policy that the terminal will be, temporarily or permanently, no longer allowed to access the telecom system domain. When the Access_Result is not_granted, the Access_Rights returned to the TA from the ADF may contain a no_of_retries field increased by one. The no_of_retries field indicates the number of unsuccessful access attempts and is used as contextual information for the next access control. The TA will invoke the operation Put(Access_Rights) on the Term_Profile object to save the updated Access_Rights. Depending on the operation which initiated the access control procedure, the TA will return the appropriate response containing an access_not_granted status.

7.4 SECURITY SERVICES RELATED TO USER MOBILITY

7.4.1 Identification

In order to support mobility, the user must have a unique identity (USI) which is distinct fo both the terminal identity and the network access point identity. A user may have several identities, for example, one or several publicly known identities such as email address and one secret and personal (Personal User Identity (PUI)) such as

login name that he uses to identify himself when accessing the telecom system domain. Each identity identifies unambiguously a unique user.

The identification of the user is incorporated in the user registration procedure defined in Chapter 6.

7.4.2 Authentication.

7.4.2.1 Information model

As stated earlier, a user can be a human being, a machine or a software process. A user as a human being imposes some additional requirements on the methods for authentication. The methods for authentication of human users must be acceptable to human users as well as being economical and safe.

A human being has the following characteristics:

- passive characteristic such as fingerprint, retinal characteristics, etc.
- limited information storage capability
- limited information exchange and processing capability

The authentication methods which fit the human user are:

1. method based on something known and simple, e.g. password
2. method based on something possessed, e.g. magnetic card or smart card. In this method the authentication is of the possessed object rather than the authentication of the holder. It may be strengthened by an authentication of the user to the card by method 1, i.e by a PIN code.
3. method based on the measurement of the passive characteristics.

The methods require different capabilities of the terminal where the authentication is to be performed. The first method requires very little or nothing from the terminal but is very weak since the risks of disclosure and replay are high. The second method requires that the terminal must have the capability to communicate with the card. This is a much stronger authentication than the above. The third method demands that the terminal must be equipped with sophisticated measurement devices but is simple for the user. This is a strong authentication method.

Authentication of machines and software processes should be based on cryptological methods in order to be safe, possibly involving a trusted third party.

When the user is allowed to move and use any terminal, the authentication mechanism used to authenticate the user at a given terminal must be adapted to the terminal type and terminal capability and to requirements specified in the user profile. An information model for user authentication is shown in Figure 7.4.

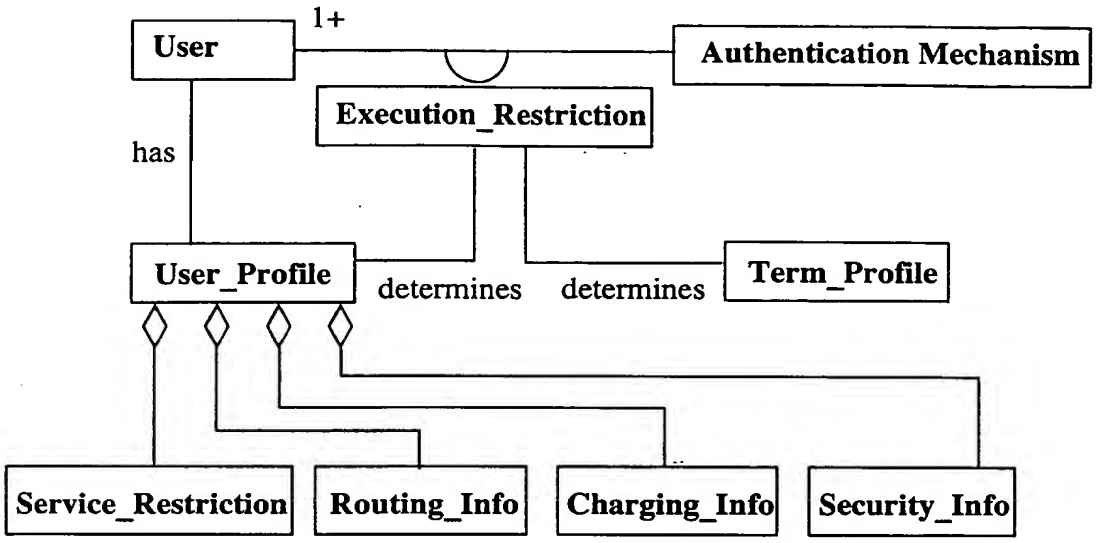


Figure 7.8 An information model for the user authentication

The relationship between the **User** and the **Authentication_Mechanism** is called **Execution_Restriction** which is determined by the **User_Profile** and the **Term_Profile**. The **User_Profile** object contains all the attributes which are specific to a user. The **User_profile** object is decomposed into four component objects: **Service_Restriction**, **Routing_Info**, **Charging_Info** and **Security_Info**. The object **Security_Info** contains necessary information for the authentication services such as authentication mechanism, password, etc. The object **Service_Restriction** contains information used for the access control of the user such as roaming restriction, credit limit, etc. The object **Routing_Info** contains information used for the delivery of services to the user. The **Charging_Info** contains information used for accounting and billing, e.g billed party, time and location dependent restrictions, etc. For more details about the **User_Profile**, see Chapter 9.

7.4.2.2 Computational model

The goal of the GMS is not to decide which authentication mechanism should be use to secure the system but to ensure flexibility and allow that different authentication mechanisms can be used in concordance with the user profile and the profile of the current terminal. The mechanism used for authentication of the user can also be used by the access control procedure to decide what access rights to be granted to the user, e.g using weak authentication may require that only a limited set of services is accessible from the terminal. Further, authentication can also be mutual, i.e the telecom system domain may also be authenticated by the user.

The type of authentication procedure used and result of the authentication must be stored so that it can be used by the access control service. Since this information is related to a specific user using a specific terminal, it is most convenient to store it in the object `User_Registration` which contains data related to the user and which is required in order to support mobility functions. For more details about the object `User_Registration`, see Chapter 6. The object `User_Registration` is expanded with a column containing security data such as `Authentication_Type`, `Authentication_Result`, `NoOfRetries`, `Terminal_Type`, etc. A row in the table contained in the `User_Registration` object contains data related to a terminal, i.e its `TA`, its `TD_UA`, its security objects, its computational objects.

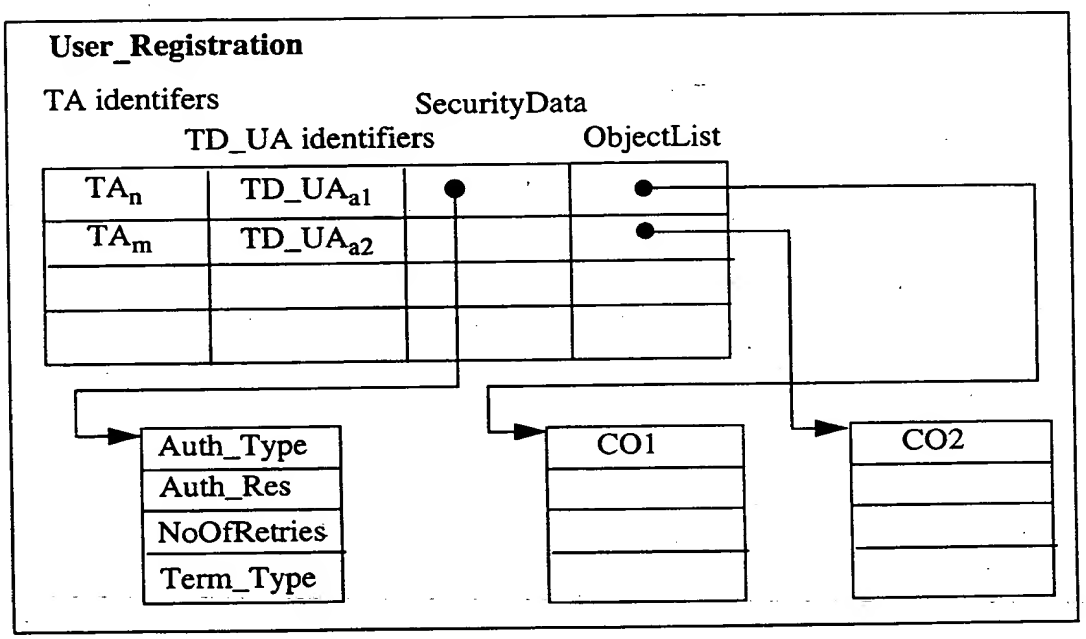


Figure 7.9 A User_registration object containing Security Data

We shall study two cases, i.e weak authentication of the user at a simple terminal and strong authentication of the user at a terminal supporting smart cards.

7.4.2.2.1 Weak authentication

The authentication of the $\texttt{User}_a$ is performed at a simple terminal by the use of PIN code. The only possible authentication mechanism in this case is that both the $\texttt{User}_a$ and his agent $\texttt{PD\_UA}_a$ knows the PIN code. The PIN code is actually stored in the `User_Profile` computational object. To assist the $\texttt{PD\_UA}_a$ in the authentication of the user, we need one object: `PD_Verifier`. It is worth noting that this object

may be of the same type as the one used for terminal authentication. Again, the common operation `Auth_Transfer` is defined for the objects **PD_UA**, **TD_UA** and **UI**. On the **PD_UA** and **TD_UA**, the operation `Auth_Result` is also defined to return the result of the verification from the Verifier. The parameters returned are the `Authentication_type`, `Authentication_Result`.

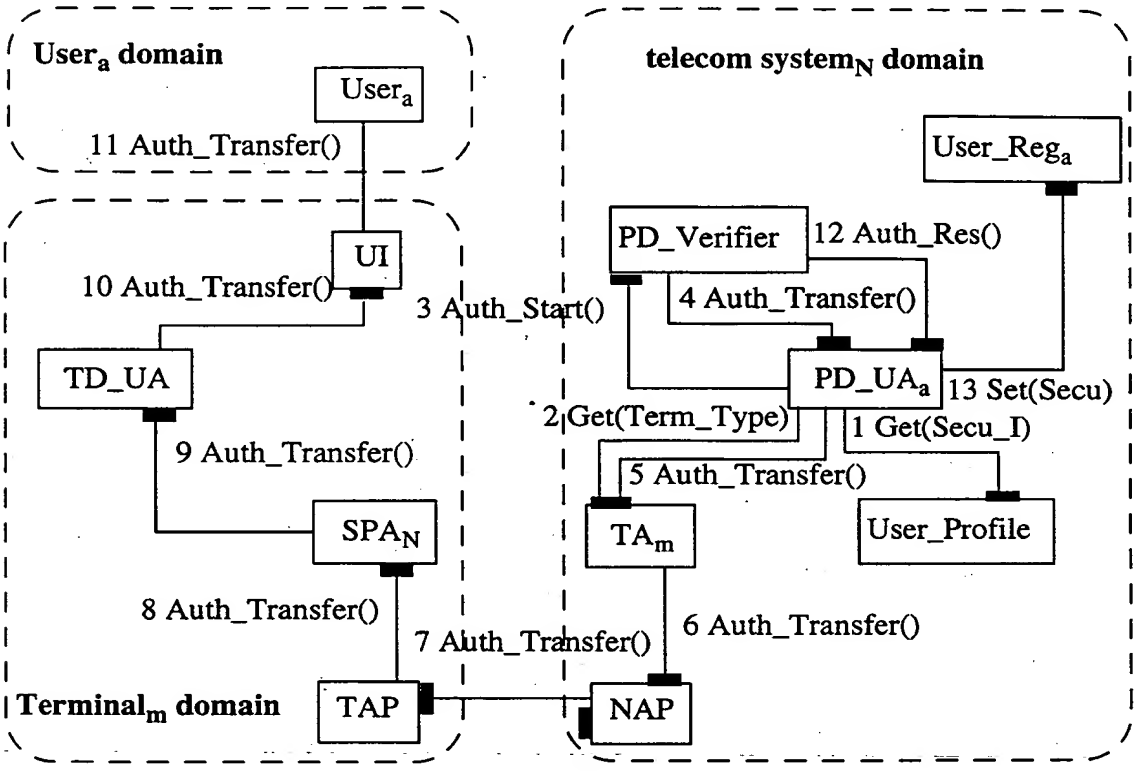


Figure 7.10 Weak authentication of the user

The authentication procedure is as follows:

1. **PD_UA$_a$** invokes the operation `Get(Security_Info)` on **User_Profile** to get necessary information to decide and initiate the authentication.

2. **PD_UA$_a$** invokes the operation `Get(Terminal_Type)` on **TA** to get the type of the current terminal which is used to decide which authentication mechanism to be used.

3. **PD_UA$_a$** assembles the obtained information into an Authentication Information (AI) parameter and uses it as input argument to invoke the operation `Auth_Start(Security_info)` on **PD_Verifier** to start the authentication.

4. From the AI received, PD_Verifier knows that an authentication based on PIN code should be used. It incorporate hence a PIN code request in the operation Auth_Transfer() which is invoked on the PD_UA_a.

5. PD_UA_a invokes Auth_Transfer()on TA.

6. TA invokes Auth_Transfer() on NAP.

7. NAP invokes Auth_Transfer() on TAP.

8. TAP invokes Auth_Transfer() on SPA.

9. SPA invokes Auth_transfer() on TD_UA_a.

10. TD_UA_a invokes Auth_Transfer()on UI.

11. UI delivers the challenge, i.e PIN code request, to the $User_a$.

The $User_a$ answers with a PIN code which is conveyed all the way back to the PD_Verifier. All the operations are however omitted in Figure 6.3 for clarity and simplicity.

12. PD_Verifier does the verification and invokes Auth_Result on PD_UA_a to pass the result (which may either be successful or unsuccessful authentication) together with the authentication mechanism used.

13. PD_UA_a invokes the operation Set(SecurityData) to update the User_Registration with the authentication result which will be used as contextual information in the access control service.

Using the same argument as in Paragraph 7.3.2.2, we then avoid that PD_UA_a is burdened with the task of taking security decisions. No matter what the result is, the PD_UA_a will proceed with the access control service of the terminal because the access is decided by the access control and not by the authentication.

The procedure is the same whether the application requiring authentication was initiated by the user or by the telecom system domain.

7.4.2.2.2 Strong authentication

The $User_a$ is now equipped with a smart card [Wol94]. In order to use a terminal, $User_a$ inserts the smart card into the terminal. Prior to any action, $User_a$ may have to authenticate himself to the card. He has for instance to enter the correct PIN code that is verified by a TD_Verifier object residing on the smart card. If the local authentication is successful, $User_a$ can now try to log into the telecom system domain. This will result into another authentication initiated by the PD_UA_a. Now, $User_a$ is represented by a TD_Claimant object on the smart card which performs the authentication for him. The authentication mechanism performed can hence be more complex and of higher class then the one we described in the previous paragraph, and may consist of a series of challenges and responses. The authentication can also be

mutual, i.e the telecom system domain is authenticated by $\texttt{User}_a$. However, the authentication procedure at the level seen from the GMS as shown in Figure 6.3, is quite similar to the one for weak authentication and it is not necessary to go through the details once more.

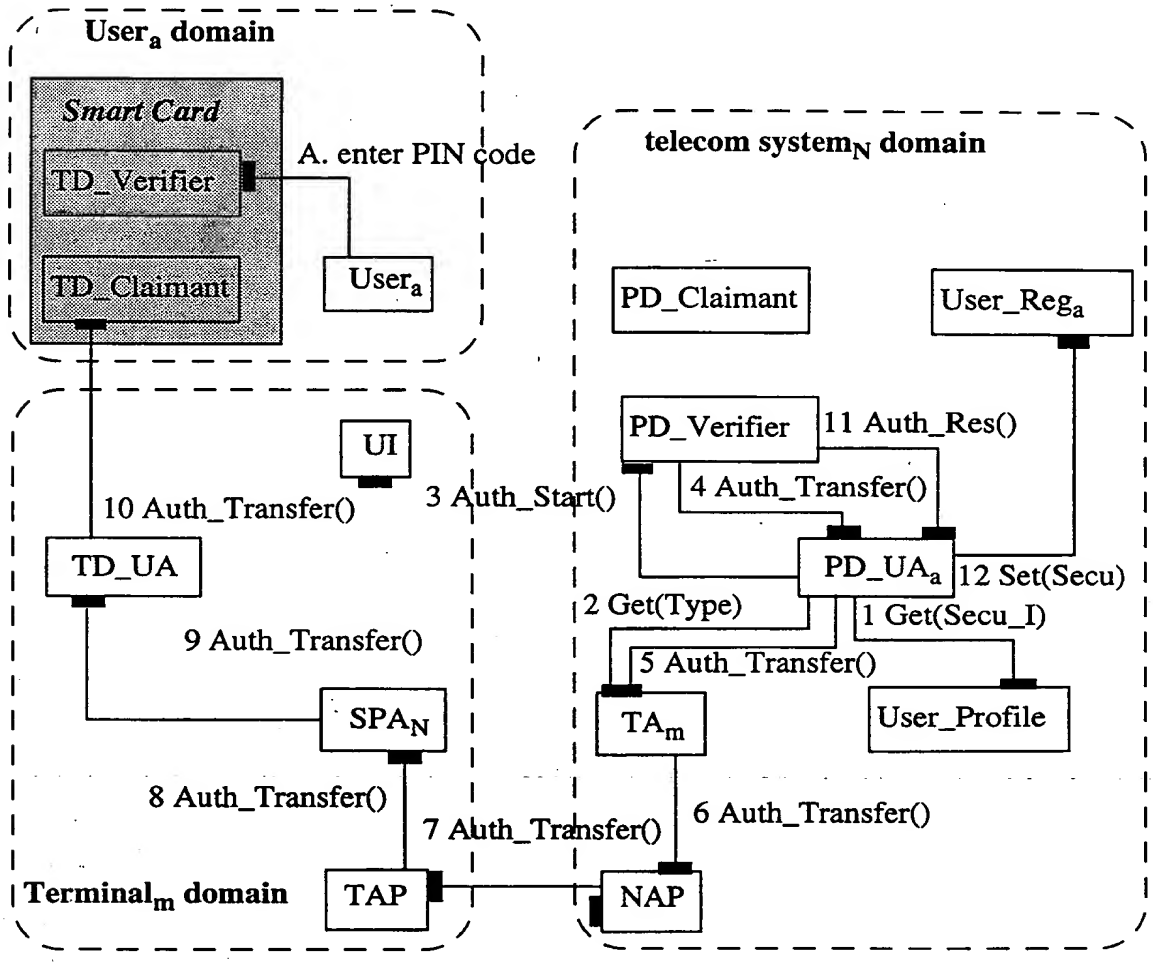


Figure 7.11 Strong authentication of the user

Note, however, that we now need additional objects `TD_Verifier`, `TD_Claimant` and `PD_Claimant` (for mutual authentication). These objects are very much similar to those used for terminal authentication.

7.4.3 Access Control

Before allowing the user to access the services offered by the telecom system domain, he is subject to three types of access control:

- access control concerning the use of the current terminal (protection of third party)
- access control concerning the access to the telecom system
- access control concerning the use of the service requested

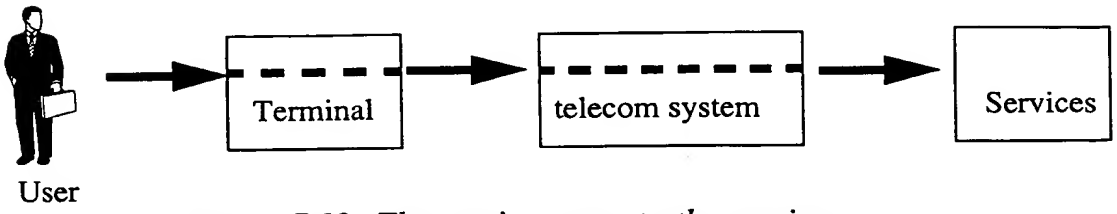


Figure 7.12 The user's access to the services

7.4.3.1 Access control for use of the current terminal

With mobility, users may make use of any existing and available terminals and network access points. However, this does not mean that the terminal owner (the third party) has to accept such actions on his terminal. He must have the rights to restrict the usage of the terminal, e.g only allowing certain users while other are prohibited from using the terminal. Of course, there are many ways to do this locally, e.g. keep the terminal in a secure place, use local password, etc. but they are cumbersome for the owner and often not secure enough. This is commonly referred as the protection of third parties [ETSa].

The information required for to carrying out the access control is contained in the **Usage_Restriction** component of the object `Term_Profile` (see Figure 7.5 in Paragraph 7.3.2.1). The attribute `All_Barring` is used to specify that only the terminal owner can use the terminal. The terminal owner may also prevent a particular user or group of users from using his terminal by specifying the attribute `Barring_List` or to allow only certain user by specifying an `Allowance_List`. Modification of the **Usage_restriction** may be provided as an application where only the owner has the right to make access. The details of such an application and the specific layout of the **Usage_Restriction** is a matter of implementation and will not be studied further in this thesis.

Let us recall from Chapter 6 that interactions of objects belonging to a $\texttt{User}_a$ at a Terminal_m always go through the $\texttt{TA}_m$object. It is therefore possible for $\texttt{TA}_m$ to assume the Access control Enforcement Function (AEF) and initiate the access control procedure for each traversing operation and stop the conveyance of the operation if the access control procedure results in a negative access decision.

This solution is however very inefficient when there are many interactions between the user domain and the telecom system domain. If the `TA` has the capability to re-

member and distinguish between the users who has been given clearance and the ones not yet controlled and thus requiring access control, then the amount of processing will be reduced considerably since the number of users at a terminal is smaller than the number of interactions..

As specified in Chapter 5, a computational object **Terminal_Data** contains information which are required to support terminal mobility. In order to support selective access control of the terminal, the object **Terminal_Data** may be expanded to contain a table of controlled and cleared users, called ClearedUserTable, as shown in Figure 7.13. The ClearedUserTable contains the references or CIIs (Computational Interface Identifier) of the **PD_UA**s whose access have been controlled.

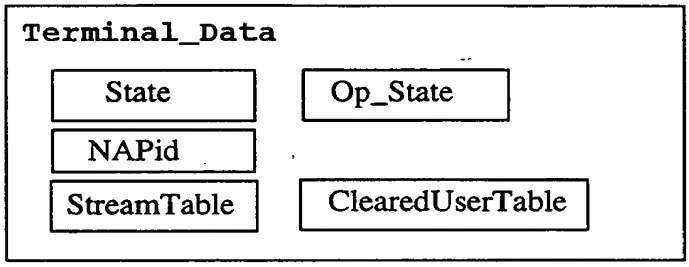


Figure 7.13 The expanded Terminal_Data object

The access control Procedure for use of the terminal is shown in Figure 7.7.

1. Every time an operation OpX arrives at the **TA**, the **TA** will check whether the CII of the originating or addressed **PD_UA** is on the ClearedUserTable or not. If it is, **TA** will do the transfer of OpX If it is not, **TA** will initiate the access control Procedure.

2. **TA** invokes Get(Usage_Restriction) on **Term_Profile** to acquire the access control Decision Information (ADI).

3. The **TA** invokes the operation Decision_Request on the **ADF** object. The arguments of this operation are the ADI obtained from the **Term_Profile**. The **ADF** makes the decision and returns the Access_Result to **TA**. The Access_result may be granted or not_granted.If the Access_Result is not_granted, **TA** returns an error message to the originator of the operation.

4. If the Access_Result is granted, **TA** invokes the operation Update(ClearedUserTable,PD_UARef) on **Terminal_Data** to register the **PD_UA** of the newly cleared user.

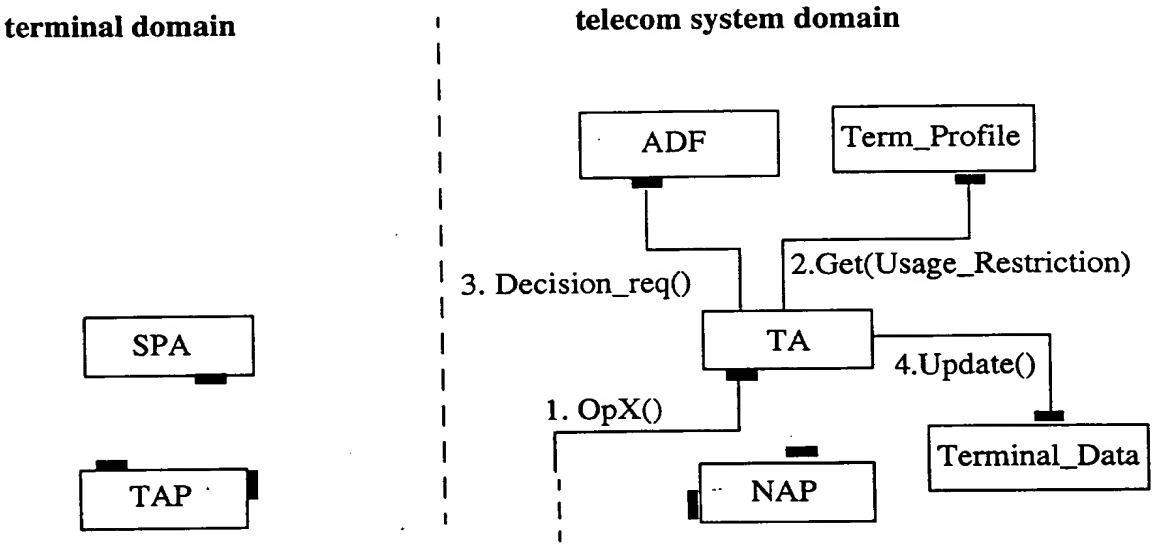


Figure 7.14 A computational model of the access control of the user for use of the terminal

One way of removing entries from `ClearedUserTable`, i.e the CII of a **PD_UA**, is to restart a timer each time that entry is acessed. If the timer times out, the entry is removed. Some entries may be permanent, i.e they are not associated with a timer.

This type of access control is only intended to other users than the terminal owner himself. In fact, the terminal owner should never be prevented to use his terminal. The access to the telecom system domain and the access to the services are different types of access controls which are applicable to all the users including the terminal owner and will be studied in the next paragraphs.

In the object `Usage_Restriction` it is therefore necessary to define an additional attribute called `NoRestr_List` containing the **PD_UA** CIIs of the users who are by default allowed to use the terminal. The CII of the terminal owner's **PD_UA** is one of them. This list must not be accessible to anyone but the telecom system domain operator itself,i.e even not to the terminal owner. However, it may be possible to define an "*emergency user*", i.e every call to an emergency number will be effectual without being checked by the access control service.

7.4.3.2 Access control for access to the telecom system domain

If the user is allowed to use the terminal, it does not necessarily mean that he is allowed to access the telecom system domain. He may be located outside the roaming area; his credit with the operator may have run out; the authentication mechanism used to authenticate him may also be too weak and he is allowed to access a limited set of services. The list of allowed services for a user at a terminal is hence equal to or smaller than the list of subscribed services. This list is stored in the

`User_Registration` object. The expanded `User_Registration` is shown in Figure 7.9. Here we have included a new column containing list of allowed services.

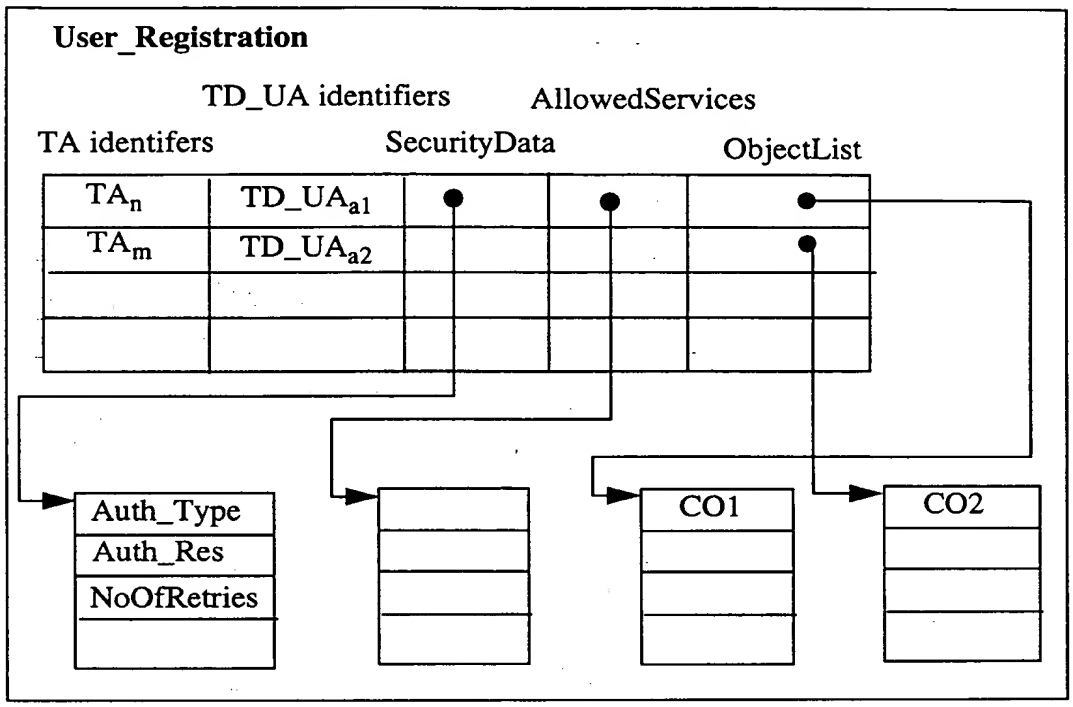


Figure 7.15 A User_registration object containing the list of allowed services

The initiator of the access control service is $\texttt{User}_a$. The target is the telecom system domain. The **AEF** is assumed by the $\texttt{PD\_UA}_a$. The **ADF** is assumed by the object **ADF**. The access of the user to the telecom system domain may be limited by some parameters such as `Roaming_Restriction`, `Credit_Limit`, `Time_Restriction`, etc. which are contained in the `Service_Restriction` attribute of the `User_Profile` object. The `Service_Restriction` attribute contains also a list of subscribed services. The use of the services in this list may be conditioned by the strength of the method used to authenticate the user, the location of the terminal, the call destination, etc. The `Service_Restriction` attribute may thus be quite complex.

A computational model of the access control service for access to the telecom system domain is shown in Figure 6.3.

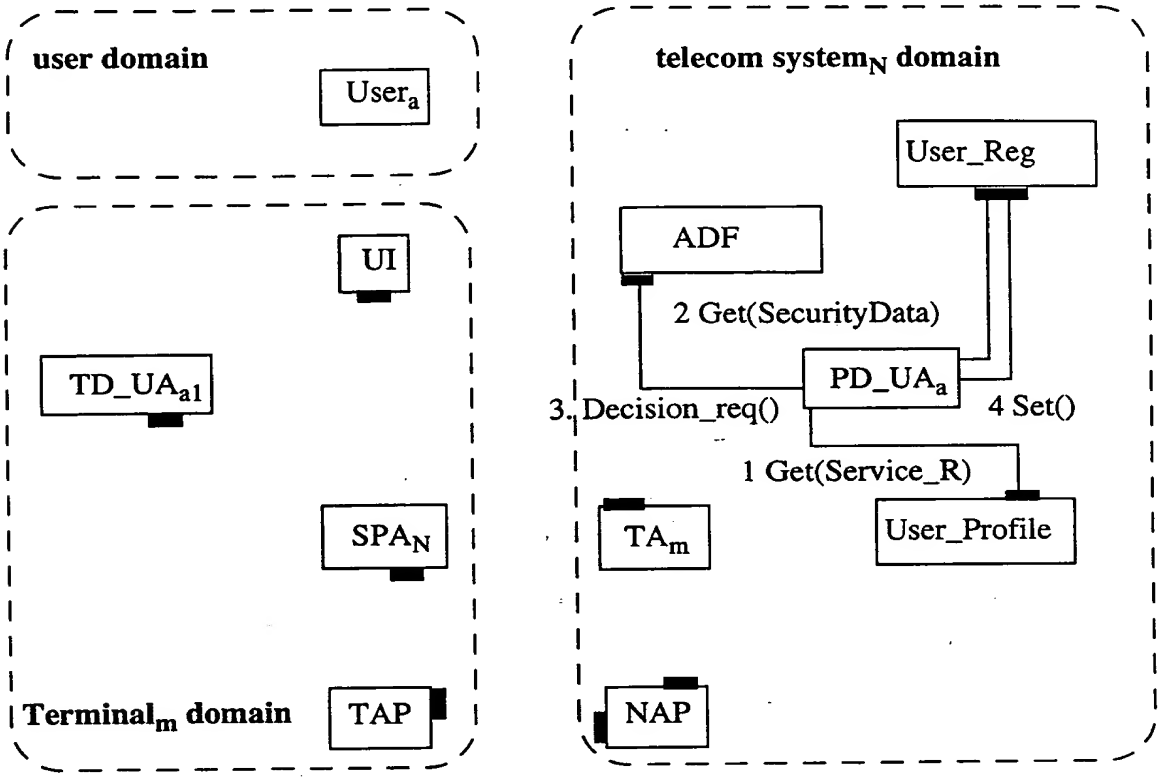


Figure 7.16 access control on the access to the telecom system

The access control procedure is as follows:

1. The **PD_UA$_a$** object invokes a `Get(Service_Restriction)` on the **User_Profile** to acquire the access control Decision Information (ADI).

2. The **PD_UA$_a$** object invokes a `Get(SecurityData)` on the **User_Registration** object to acquire the contextual information (result from the authentication service).

3. The **PD_UA$_a$** object invokes the operation `Decision_Request` on the **ADF** object. The arguments of this operation are the ADI obtained from **User_Profile** and the contextual information obtained from **User_Registration**.

The **ADF** may use the access control services offered by the platform or a security system to obtain further contextual information such as time, system status, etc. and the access control policy rules. The **ADF** makes the decision and returns the `Access_Result` to **PD_UA$_a$** together with `SecurityData` and `AllowedServices`.

The Access_result may be granted, not_granted or suspended. If the Access_Result is Suspended, depending on the access control Policy the terminal will be, temporarily or permanently no longer allowed to access the telecom system domain.

If the Access_Result is not_granted, the SecurityData returned to the PD_UA_a from the ADF will contain a NoOfRetries field increased by one. The NoOfRetries field indicates the number of unsuccessful access attempts and is used as contextual information for the next access control service. The PD_UA_a will invoke the operation Set(SecurityData) on the User_Registration object to save the updated SecurityData. Depending on the operation which initiated the access control procedure, the PD_UA_a will return the appropriate response containing a not_granted status.

When the Access_Result is granted, the AllowedServices containing an updated list of allowed services is returned to the PD_UA_a. The PD_UA_a will invoke the operation Set(AllowedServices) on the User_Registration object to save the updated AllowedServices. Depending on the operation which initiated the access control procedure, the PD_UA_a will return the appropriate response containing a granted status.

With the accomplishment of the access control service between the user and the telecom system domain, the access session is completed. The concept of access session is described thoroughly in Chapter 9. The user can now request the wanted service and is hence subject to the access control for the requested service.

7.4.3.3 Access control for the requested service

There are two types of services, outgoing and incoming (see Chapter 9). Outgoing services are initiated by the user himself while incoming services are delivered to him by other users or applications.

For outgoing services, the initiator of the access control service is $User_a$. For incoming services the initiator is some other user or application. The target is the requested service. The AEF is assumed by the PD_UA_a. The ADF is assumed by the object ADF. The access of the user to the requested service is limited by the information contained in the AllowedService list of the User_Registration object. Another restriction originates from the Usage_Restriction contained in the object Terminal_Data and set by the terminal owner. The terminal owner may allow only one or both of the two service types to be performed on his terminal The attributes OutBarring and InBarring of the Usage_Restriction is used to specify, respectively, the users who are not allowed to use outgoing services and incoming services on the terminal (or who are allowed).

The access control procedure is as follows:

1. The PD_UA_a object receives a `ServiceReq(ServId)` from either the user or an application.

3. The PD_UA_a object invokes a `Get(Usage_Restriction)` on the `TA`.

3. The PD_UA_a object invokes a `Get(AllowedService)` on the `User_Registration`.

2. The PD_UA_a object invokes the operation `Decision_Request` on the `ADF` object. The arguments of this operation are the ADI obtained from the `User_Registration object` and the `TA`.

The `ADF` makes the decision and returns the `Access_Result` to PD_UA_a. The `Access_result` may be `granted or not_granted`. Depending on the operation which initiated the access control procedure, the PD_UA_a will return the appropriate response to the requester. The access control on the requested service is shown in Figure 7.17

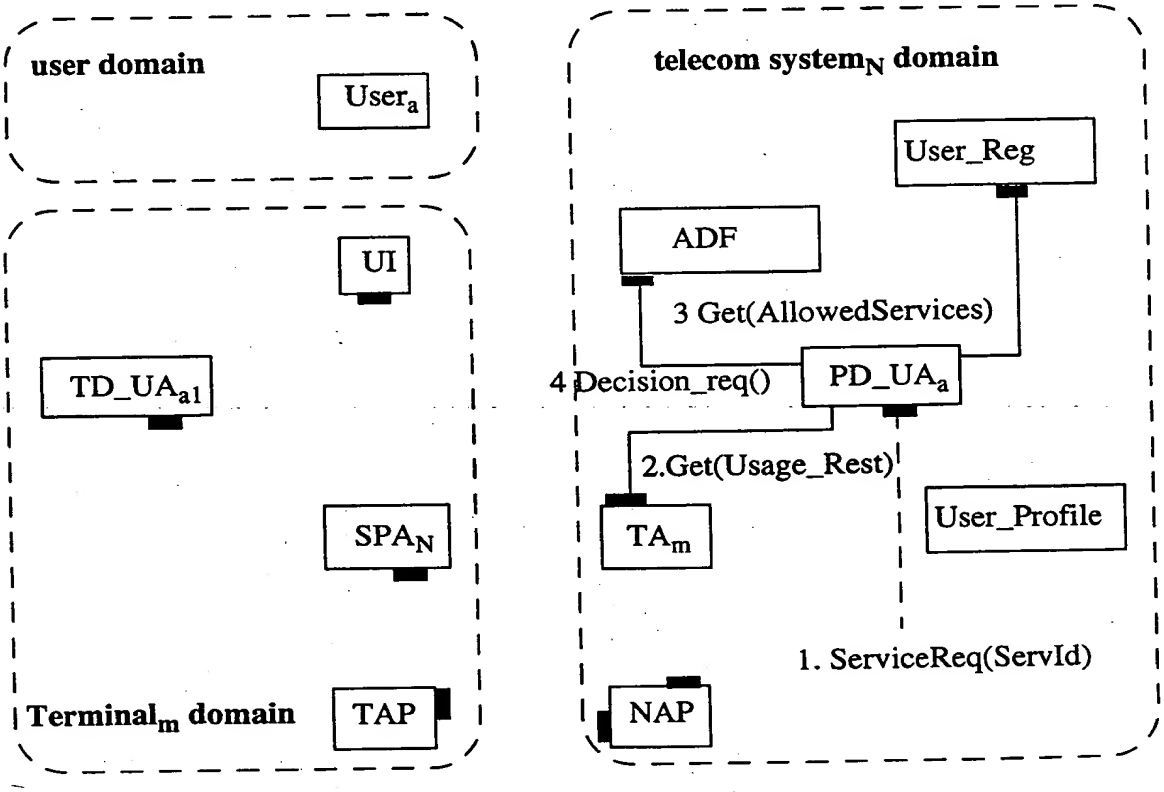


Figure 7.17 Access Control on the requested service

7.5 CONCLUSION

In this chapter we have examined how the security aspects and services are affected by mobility. To support mobility, additional security functions must be introduced in the GMS. These functions are assured by the following new computational objects:

- For authentication:
  - `TD_Claimant`
  - `TD_Verifier`
  - `PD_Claimant`
  - `PD_Verifier`
- For access control:
  - `ADF` (Access control Decision Function)

These new objects and their interactions with security services constitutes the interface of the GMS with a Generic Security System.

We have also observed that in order to accomplish their mission, the introduced security objects need both assistance and information from some GMS objects. These GMS objects must therefore be enhanced with functionality in addition to that described in Chapter 5 and Chapter 6. The GMS information objects must be expanded with additional attributes while the GMS computational objects must be equipped with new operations. Since information objects are usually incorporated inside a computational object, we can say that security concerns the following computational objects:

In the terminal domain

- `SPA`
- `TD_UA`

In the telecom system domain:

- `Term_Profile`
- `PD_UA`
- `User_Profile`
- `User_Registration`

8

Making mobility transparent

8.1 INTRODUCTION

In the previous chapters we have identified the functions necessary to support terminal and user mobility. We are now going to study the different techniques to make these functions transparent to the applications. Ideally, the application designer should not have to be concerned about mobility and, instead can concentrate on solving the problems which are related to the application domain. In other words, the mobility related functions should be incorporated into the system and kept hidden from the application. When this goal is achieved, the design and implementation of mobile applications are no more complex than the ones of "fixed" applications.

8.2 THE ODP/TINA CONCEPTS AND THE SYSTEM DEVELOPMENT PROCESS

Before studying how mobility can be made transparent, it is necessary to consider

- which are the defined distribution transparencies
- what does they mean to the application designers
- when and in which phases is distribution transparent
- when is it visible,
- how transparencies can make the job easier for designers

8.2.1 Review of ODP/TINA concepts

The objective of ODP is the development of standards that allow the benefits of distribution of information processing services to be realized in an environment of heterogeneous IT resources and multiple organizational domains. This comprises the provi-

sion of infrastructure components and functions that accommodate difficulties inherent in the design and programming of distributed systems[ITUa].

To achieve the objective, the RM-ODP (Reference Model for Open Distributed Processing) is developed to provide a framework for system specification and the corresponding infrastructure components.

The RM-ODP defines

- a division of an ODP system specification into five *viewpoints*, in order to simplify the description of complex systems;
- a set of general concepts for the expression of those viewpoints specifications;
- a model for an infrastructure supporting, through the provision of *distribution transparencies*, the general concepts that it offers as specification tools;
- principles for assessing *conformance* for ODP systems.

The first three viewpoints, namely Enterprise, Information and Computational, specify the ODP system in distribution transparent terms, i.e the models are constructed assuming certain distribution transparencies. The mechanism to provide the identified transparencies are defined in the Engineering viewpoints. Distribution is accordingly kept hidden in the three first viewpoints.

8.2.2 Mapping to the development process

The RM-ODP defines only the framework to specify a distributed system but it does not prescribe a methodology. There is a clear distinction between framework or method and methodology. As stated in [Boo91, p. 18-19], a framework or method is a disciplined process for generating a set of models that describe various aspects of a software system under development, using some well-defined notation. A methodology is a collection of methods applied across the software development life cycle and unified by some general, philosophical approach.

The RM-ODP does not dictate how the viewpoints should be applied in the different phases of the development process of a distributed system. The development process is usually iterative and earlier phases must be revisited in order to solve newly encountered problems. There are therefore different methodologies defining the use of the viewpoints in the development phases. We choose however to consider the methodology proposed in [Aud96] as depicted in Figure 8.1.

In this methodology, the distribution aspects are not visible in the requirements and functional specification phases but must be taken into consideration in the design and implementation phases. From this point of view, the RM-ODP alleviates the construction of distributed systems in the sense that application designers does not have to be concerned with distribution and can concentrate on their specific problems in the requirements and functional specification phases. However, the RM-ODP by itself is not sufficient to make the design and implementation of distributed systems easier. In these two phases the existence of an infrastructure or platform supporting the re-

quired distribution transparencies and the availability of development tools play a crucial role. The merits of the RM-ODP are indeed through its huge contributions to the design and implementation of such platforms and development tools.

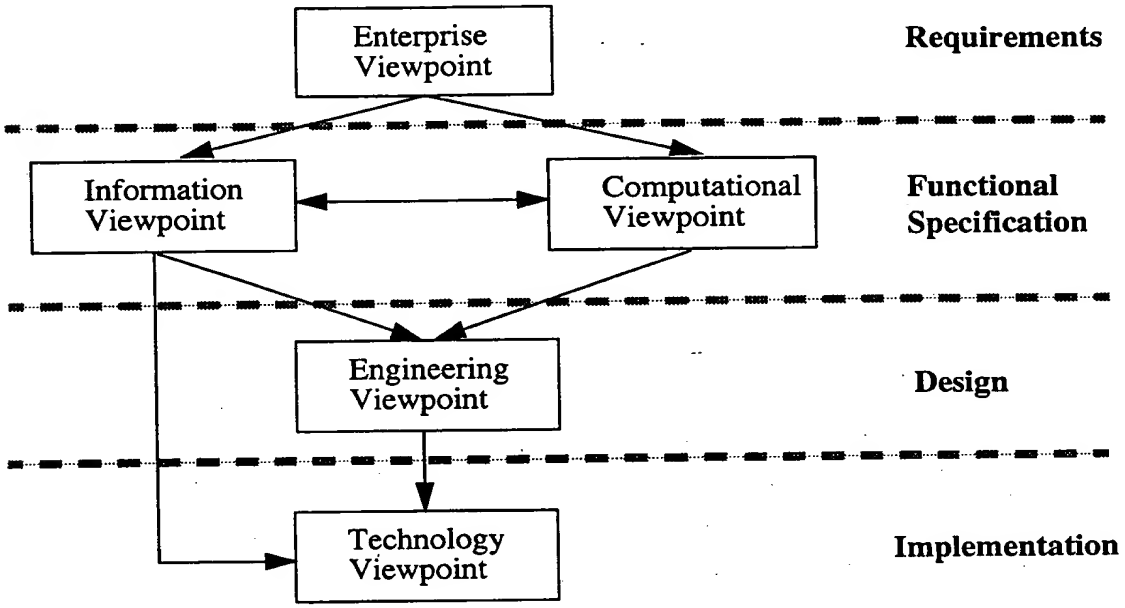


Figure 8.1 The role of the different viewpoints in the development process

8.2.3 Example of system development

In order to clarify our point, let us consider the development of an arbitrary system S.

The requirement phase is assured by the enterprise model of S which consists of enterprise objects that assure different roles (also called actors) and expresses the objectives and policy constraints on the system.

The functional specification phase is accomplished through the information model and the computational model. The computational model decomposes S into computational objects performing individual functions and interacting at well-defined interfaces. The computational model forms the basis for the decision on how to distribute the jobs to be done. It specifies in fact the "maximum distribution" case where each object is residing in a node (This will be constrained later in the engineering model). The computational model is however not concerned with the mechanism necessary to support distribution. The information model describes the information stored and manipulated by S. The information objects are contained in one or more computational objects. They may be exchanged between computational objects. They may be volatile or persistent. The information model forms also the basis to require the persist-

ence and transaction transparencies in the engineering model. It can be used later in the implementation phase to realize the necessary data storage (e.g designing database schemas).

In the design phase, decisions must be made concerning how computational objects should be mapped into the engineering objects and how they should be distributed. The computational objects are hence grouped into cluster, capsule and node. There are different communication mechanisms to support object interaction depending on the form of grouping:

- If two objects are within the same cluster, the communications between them may be direct and highly optimized, e.g method calls.
- If two objects are within different clusters, the communication between them are supported by a channel concept which consists of the three engineering objects stubs, binder and protocol. Stubs are concerned with the information conveyed in an interaction, binders are concerned with maintaining the association between the set of basic engineering objects linked by the channel, and protocol objects manage the actual communication.
- If two objects are within different technical or administrative boundaries (or domains), an interceptor must be introduced in the channel in order to perform additional checks or transformation to match the requirements in the two domains.

Figure 8.2 shows an example of a system S which consists of four computational objects that are mapped into four basic engineering objects O1, O2, O3 and O4. The distribution scheme chosen for system S is: O1 and O2 are within cluster 1, O3 is within in cluster 2, O4 is within cluster 3. Cluster 1 and 2 are within capsule a, node A and domain X. Cluster 3 is within capsule b, node B and domain Y.

From the chosen distribution the necessary communication between objects can be deduced. Communications between O1 and O2 can be done directly because they are in the same cluster. Communications between O2 and O3 must be done through a channel consisting of stubs, binders and protocols because they are in different clusters. Communications between O3 and O4 also go through a channel but this time in addition to stubs, binders and protocol, there is a need for an interceptor because O3 and O4 belong to different domains.

Suppose now that the infrastructure does not support any distribution transparency at all. Then, before proceeding to the implementation phase, the application designer has to design for each application all the engineering objects stub, binder, protocol and interceptor, plus all other system management functions such as node management, capsule management, etc.

The implementation phase of the system S will then consist of the implementation of the basic engineering objects O1, O2, O3 and O4 and the implementation of the engineering objects stub, binder, protocol and interceptor, and all other platform functions. And this constitutes a tremendous job.

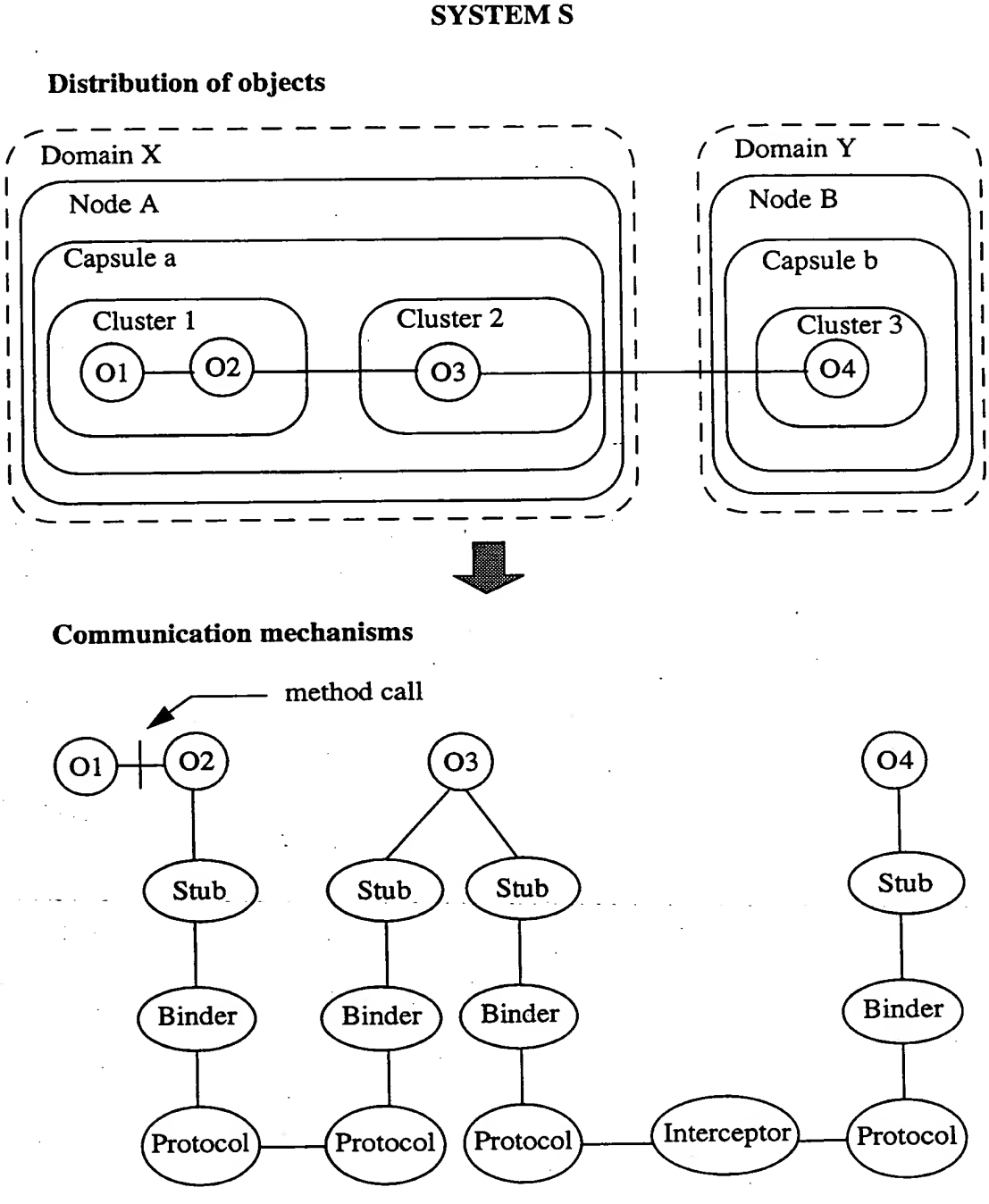


Figure 8.2 Engineering model of system S

Fortunately, the platform functions need only to be built once if we apply the ODP framework. The stubs, binders, etc. can be designed and implemented in such a way that they can be generated mechanically and used for all basic engineering objects. The infrastructure supporting access and location transparencies will usually offer

tools to the automatic generation of these engineering objects. Some engineering objects such as stub will usually but not necessarily, be offered as programming codes which can be included and compiled together with application object. Other objects such as binder and protocol can be dynamically created at run time at the establishment of the channel.

With such an infrastructure, the design and implementation phase of distributed systems are no more complex than the one of centralized systems since the application designer just have to programme his application code.

One important issue to notice is that the distribution scheme once chosen will become the most distributed configuration that the system is allowed to have. Objects grouped within the same cluster cannot be separated and placed in different clusters. The same applies for capsule, node and domains. The restrictions are purely technical and due to the fact that the channels (stubs, binder, etc.) are defined and cannot be changed.

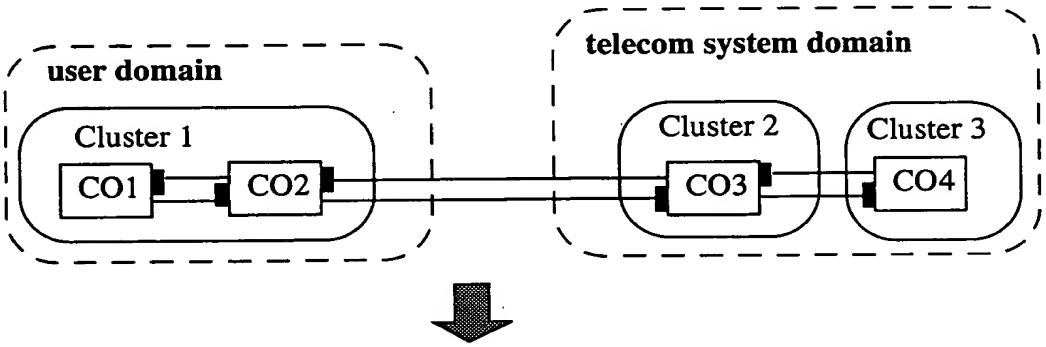
On the other hand, changes in the opposite direction, i.e changes to a less distributed configuration, are allowed but may not be very efficient since unnecessary communication channels are used for communications between objects instead of direct method calls.

8.3 THE IN-LINE ALTERNATIVE

The most straightforward alternative to make mobility transparent to the application designers at the requirements and functional specification phases, is to integrate the Generic Mobility System totally into the infrastructure or platform. All the computational objects defined in the GMS will not be mapped to basic engineering objects (BEO) but to engineering objects (EO) and will hence not be visible in the computational model of the application. In fact, the whole GMS constitutes the engineering object interceptor, standing at the boundary of two domains: the terminal domain and the telecom system domain. This solution is also called **in-line** because the support of mobility is introduced directly embedded into the platform.

In Figure 8.3, we show as example an application consisting of four computational objects CO1, CO2, CO3 and CO4. CO1 and CO2 are grouped together into cluster 1 and belong to the user domain. CO3 and CO4 belong to the telecom system domain but are assigned to clusters 2 and 3.

Computational Model of the application



Engineering Model of the application

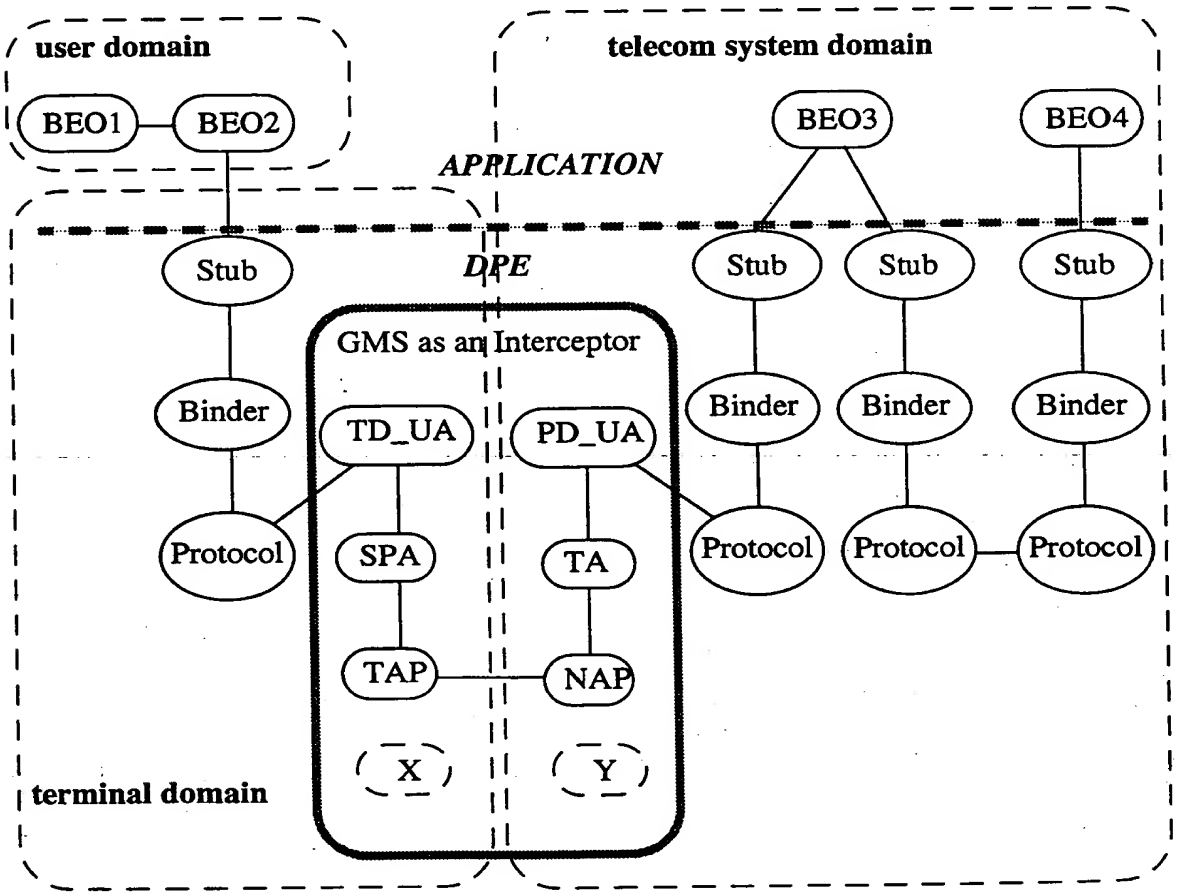


Figure 8.3 The In-line alternative

From the given computational model, an engineering model can be developed. Each Computational Object (CO) is mapped to a Basic Engineering Object (BEO). Of course, a CO can also be mapped to several BEOs but for the sake of simplicity we do not choose this mapping here. Since CO1 and CO2 are within the same cluster, interactions between them can be done directly as method calls. Communications between CO3 and CO4 go through a channel consisting of stubs, binders and protocols since they belong to different clusters. Communications between CO2 and CO3 go also though a channel because CO2 and CO3 reside in different clusters, but the channel will now contain an interceptor in addition to the usual objects because it crosses the boundary between two domains (terminal domain and telecom system domain). However, the interceptor is not visible for the application designer and does not have any consequence for the application objects. It is worth noting that the application designer does not need to think about the terminal domain in the computational model (upper part of Figure 8.3). In the engineering model (lower part of Figure 8.3) there is also drawn a borderline between the application layer and the DPE/platform layer. This borderline is in fact not a real and existing border but just a virtual one. In showing that, we aim to visualize that the GMS is fully integrated in the DPE.

The introduced interceptor is nothing else but our Generic Mobility System (GMS). In Figure 8.3, we show only the fundamental objects of the GMS. The objects X and Y represent all other objects which are identified and defined in previous chapters.

If the GMS is implemented and fully integrated into the platform and there exist development tools able to generate channels containing the GMS as interceptor, then the design and implementation of mobile applications are no more complex than the ones of fixed applications. From the computational model, in addition to the usual grouping of objects into clusters, capsules, etc., the application designer has to decide whether an object belongs to the user domain or the telecom system domain. Once the decisions are made, the necessary objects such as stubs can be generated for all the applications objects. The application designer just need to implement the programming codes which are specific to his application domain.

The presented alternative has many advantages. It succeeds in making mobility transparent to the applications. It can be optimized to become very efficient. It is also conformed to the RM-ODP specifications concerning the channel concept.

There are however some disadvantages. From the application point of view, there may be a lack of flexibility in the sense that objects, once allocated, are not allowed to migrate from the user domain, i.e from the mobile side of the system to the telecom system domain or vice versa. The boundary between the user domain and the telecom system domain constitutes a very rigid borderline which may not be wanted. From the platform point of view and also from the GMS point of view, it may not be desirable to tie closely the GMS to the platform. The platform should be general enough to fit most distributed systems without major modifications, while the GMS, as its name tells, should be generic and easy to be customized for a specific system. It is therefore desirable that the construction of the platform and the GMS could be done independently and by different parties.

8.4 THE BROKER ALTERNATIVE

8.4.1 The broker concept

In this alternative, we introduce and use a notion called **broker** defined in "Understanding Corba" [OPR96]. A broker is a specialized server object acting as intermediary between one or more client objects and one or more server objects. A broker represents a client when the client makes a request to a server and represents a server when a server makes a response to a client.

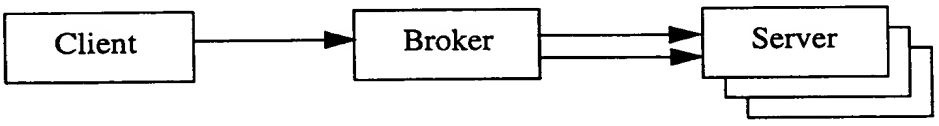


Figure 8.4 Using broker in client /server interactions

There are defined two brokering schemes:

1. **Introduction brokering**
In an introduction brokering scheme active servers register with the broker to make themselves known in the system. The client can then send a message to the broker asking for a server that can perform some task. The introduction broker selects a server to perform the task and returns information to the client accessing that server. The broker introduces the server to the client, so that the client and the server can contact each other directly. This brokering scheme corresponds the RM-ODP trading function [ITUc], [Aud96], [TIN95a].

2. **Routing brokering**
In a routing brokering scheme, the clients sends a message to the broker asking for a server that can perform some task. The routing broker then selects a server to perform the task and sends the client's request to that server. The broker handles all communications; no direct communication between the client and the server ever takes place.

8.4.2 User agent as routing broker

In the second solution we adopt the concept of routing brokering with a minor modification. The client send a message to the routing broker not asking for any server able to perform the task but asking for a definite server to perform this task. The broker locates the server and sends the client's request to that server.

Since an object may play both the roles of client and server, we use a model of two cascading brokers. The user agents in the terminal domain and the telecom system domain, respectively, TD_UA and PD_UA will assume the roles of these brokers.

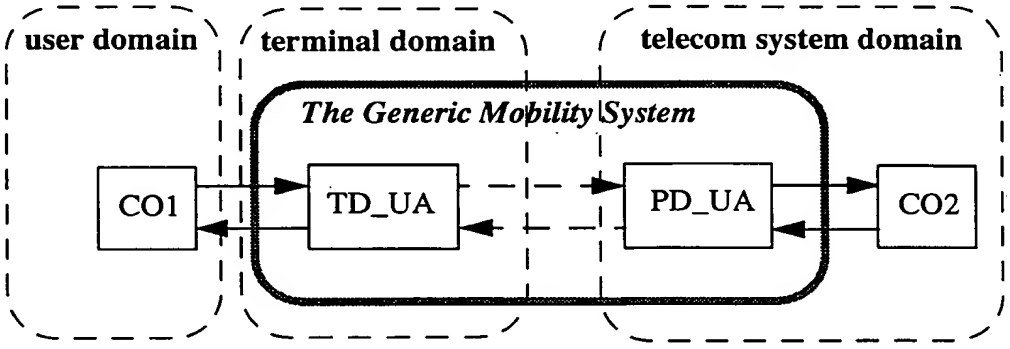


Figure 8.5 The user agents acting as routing brokers

The Figure 8.5 shows how an object CO1 belonging to the user domain interacts with an object CO2 belonging the telecom system domain. Interactions enter the GMS at one agent and exit at the other. The role of the agents are identical for interactions in the opposite direction and must hence contain the same functionality. It is worth noting that interactions go through the terminal domain which does not need to be visible until now. As described earlier, the GMS contains many other objects that interactions have to traverse, but we have deliberately omitted them in Figure 8.5 in order to preserve clarity.

Let us now examine the interfaces between the user domain's objects and the GMS. The TD_UA and the PD_UA support both the same interface type containing an "invoke type" operation. This operation originates from the same concept as the one used in the Dynamic Invocation Interface (DII) defined in CORBA [OPR96], [Objа], [BN95], [OHJ96]. This concept allows request to be built and invoked dynamically by client objects. The client object needs to know interface-related information only at the invocation time. No knowledge is necessary at compilation time. The `Invoke` is composed of an object name (or identifier), an operation name, and a parameter list for the invoked operation.

An IDL (Interface Definition Language) [Objа] of such `Invoke` operation is given in Figure 8.6.

The **`name`** is the name or identifier of the object where the call is to be made. The **`ctx`** contains information about the client object, environment, or circumstances of a request that are convenient to pass as parameters. The **`operation`** is the same operation identifier that is specified in the interface definition of the server object. The **`arg_list`** contains a list of arguments for the operation. The operation result will be placed in the **`result`** argument after the invocation completes.

```
Interface MobInvocation {
        Invoke ( in Object target;          //object to be called
                 in Context ctx;            //context of op
                 in Identifier operation;   //intented operation
                 in NVList arg_list;        //args to op
                 inout NamedValue result    //operation results
        );
};
```

Figure 8.6 IDL specification of the MobInvocation interface

The application designer can decide how a client object can call the operation **Invoke** of the user agent. If calls are to be made in a static way then the stub generated from the interface MobInvocation must be included in the client object code at compilation time. If calls are to be made dynamically, e.g through the Dynamic Invocation Interface (DII) [Objb], than the information about the interface MobInvocation is only needed at run time but not at compilation time.

After entering the GMS at the first user agent and traversing many other objects as described in previous chapters, the invocation arrives at the second user agent. In order to avoid a static binding between the user agent and an arbitrary object, and allowing independent development, the DII is again used. Hence, the user agents does not need any interface information of the server object at compilation time.

In this alternative, the GMS is introduced in the system architecture as a functional layer between the application layer and the DPE layer. It is worth noting that the mentioned layer separation is functional and not hierarchical in the sense that every layers can use services offered by other layers. Assuming that the interfaces between layers are well defined, the inside structure of a layer is not relevant to other layers and can be modified without any serious implications.

Let us now return to the same example as in the previous paragraph with four computational objects CO1, CO2, CO3 and CO4. As shown in Figure 8.3, we start with the same computational model but an intermediary model called derived computational model is introduced in the transition from the computational model to the engineering model. This intermediary model is used to map all interactions traversing the boundary between the user domain and the telecom system domain to interactions with the respective broker, i.e user agent. From the derived computational model, an engineering model can be elaborated and engineering objects such as stubs can be mechanically generated.

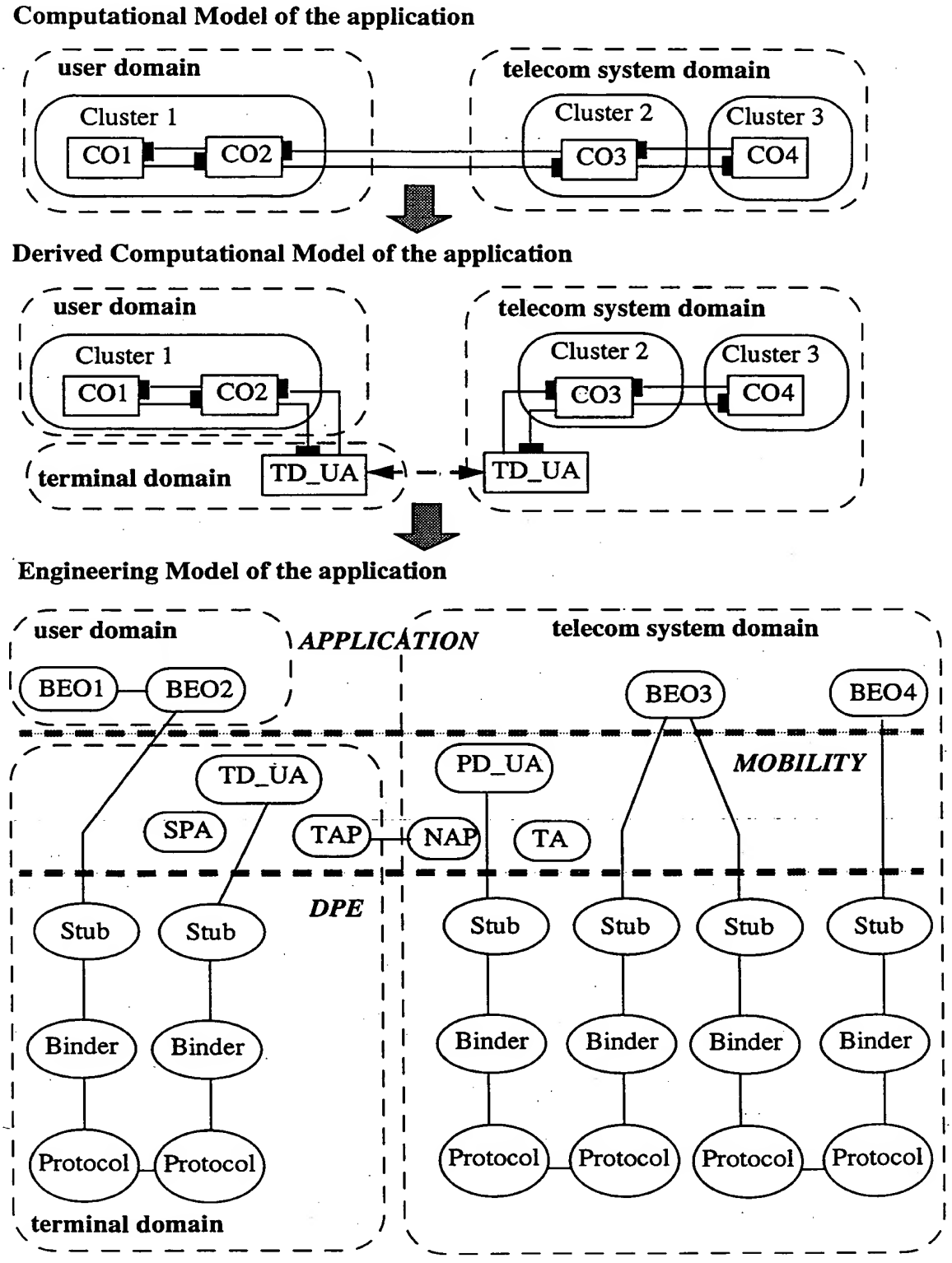


Figure 8.7 The broker alternative

This alternative has several merits. First, the mobility transparency is still preserved at the computational viewpoint. Although an additional model is necessary in the transition to the engineering viewpoint, it does not constitute a major obstacle since the derived computational model can be easily and automatically derived from the computational model.

By separating the GMS from the DPE and defining an additional functional layer for it, greater level of flexibility is achieved. Each of the three layers DPE, mobility and applications can be developed, modified or customized independently as long as the interfaces between them remain unchanged. Another advantage of this alternative is that it does not demand any particular function or feature requirements on the DPE and can be realized right away.

However, there is still one major limitation. As with the previous alternative, migration of objects across domain boundaries is not allowed. Once allocate to a domain, an object is not allowed to move since interactions across domain boundaries are statically mapped to interactions with the user agents. Objects originally designed without inter-domain interaction capability can not later on communicate with objects in other domains without modification and re-compilation.

8.5 THE PROXY ALTERNATIVE

This alternative aims to enhance further the flexibility offered by the previous alternative by loosening the closed binding between the application and the GMS. This is achieved by introducing **proxy objects**. The notion of proxy is quite similar to the one of agent. They both represent or act on behalf of another entity in certain circumstance. The difference is minimal and depends on the context. However, a proxy acts on behalf of an entity in a transparent way, i.e when interacting with a proxy, an entity cannot tell whether it is dealing with the real counterpart or with its proxy. An agent, on the other hand, is a separate entity with its own identity and the entities communicating with it are fully aware that they are dealing with an agent and not the real counterpart.

Every object requiring inter-domain interactions is represented by a proxy object located in the same domain as the object interacting with it. When interacting with an object in a different domain, the object is actually interacting with the local proxy object of the required object of the other domain. The proxy will marshall the operation invocation into the invoke operation (as defined in the previous paragraph) of the respective user agent. If interactions are to be in both directions, we need a symmetrical constellation as shown in Figure 8.8. In this example an object CO1 in the user domain wants to invoke an operation `OpX()` on an object CO2 in the telecom system domain. CO1 is actually invoking `OpX` on the proxy of CO2, namely PCO2. PCO2 will marshall `OpX()` into the `Invoke` operation of TD_UA. This operation will be conveyed through the GMS and arrives at the PD_UA which interprets and invokes `OpX()` on CO2. The result will be conveyed all the way back to CO1. It is worth noting that only PCO2, the proxy of CO2, is involved in the invocation of oper-

ations on CO2 and PCO1, the proxy of CO1 is left outside in this case. The opposite situation will occur when CO2 calls operation on CO1. There is here a difference between proxies and user agents. As shown earlier, the user agents always operate in pairs, one as the entrance door of GMS and one as the exit.

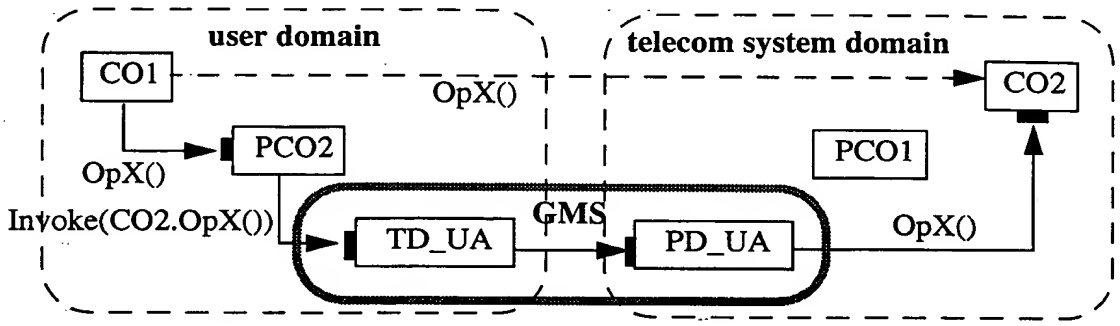
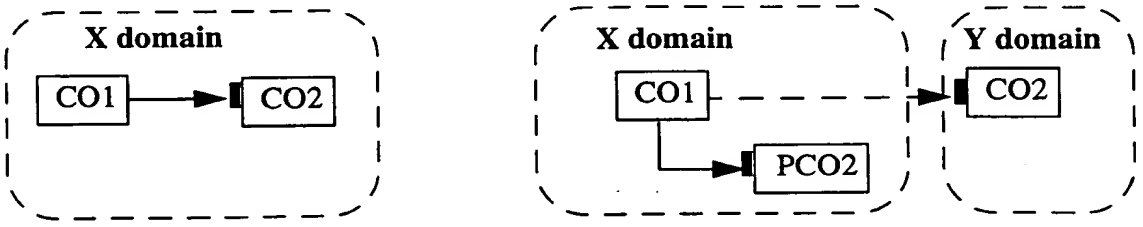


Figure 8.8 Using proxies to untie the binding between the application and the GMS

It is observed that PCO2 has an interface which has the same signature as the one of CO2. All operations defined for CO2 are defined for PCO2. However, all the operations of PCO2 will perform exactly the same job which is to marshall the initial operation and to call the `Invoke` operation of the TD_UA object.

CO1 may invoke the operation on CO2 in a static way or through the Dynamic Invocation Interface (DII). In the first case, the information about the interface of CO2 must be defined and available at compilation time. In the second case, this information is only needed at run time.

Until this stage, the level of flexibility achieved is not higher than in the first alternative. In fact, the flexibility relies on how the proxies are introduced and created. If the introduction and creation of necessary proxies are based on the domain grouping of objects and decided at compilation time, then we return to a solution quite similar to the first alternative. The proxy must somehow be created according to the communications needs and dynamically at run-time to achieve higher level of flexibility.



a) In same domain

b) In different domains

Figure 8.9 Possible configurations at run-time

Figure 8.9 shows two run-time configurations. By domain, it is meant either user domain or telecom system domain. If at run time CO1 and CO2 are within the same domains, then interactions are done through usual channels assuming of course that the access and location transparencies are supported within the domain. On the other hand, if they are in different domains, e.g one object in the user domain and the other in the telecom system domain, then a proxy is created to transfer all the interactions to the GMS which is responsible to deliver them to the other side. Neither CO1 nor CO2 need to be aware that the proxy PCO2 exists. CO1 does not distinguish between PCO2 and CO2 when it is invoking operations. CO2 does not distinguish between PCO2 and CO1 when it is invoked. It should also be possible to design and implement PCO2 independently of both CO1 and CO2. It should also be possible to create proxies dynamically at run-time when interactions across domain boundary are requested.

To fulfil the requirements mentioned above, we adopt the idea presented by OMG in the Dynamic Skeleton Interface (DSI), [Objb], [Sun94], [IT94], [orfali96:do_gui], [BN95]. The purpose for the introduction of the DSI is to enable the building of bridges interconnecting multiple, heterogeneous CORBA-compliant ORBs. However, the DSI is described in CORBA terminology and in CORBA-compliant manner which is not always consistent with the ODP/TINA concepts. We shall attempt here to map all the ideas to the ODP world and give a description which fits better with our context.

The idea is to introduce an object type called **Dynamic Object (DO)**. All the proxies will be of type DO. A proxy, i.e an object (instance) of type DO is instantiated from an object template called **Dynamic Object Implementation (DOI)**. A proxy is hence semantically separated from the object it represents. The unique relationship which remains between them is ***Represent***, i.e a proxy represents one and only one object. A proxy shall be deleted when the represented object terminates. An object can, however, be represented by multiple proxies, e.g one proxy for each client object. An information model is given in Figure 8.10.

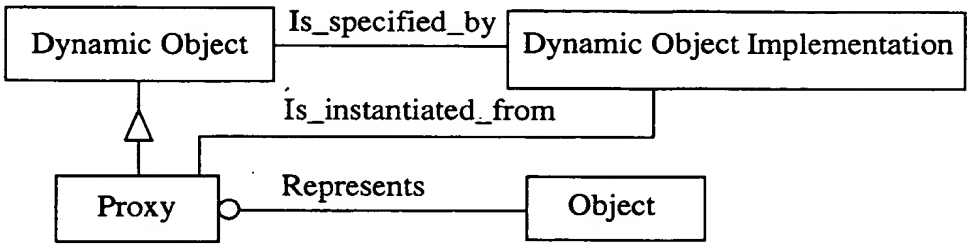


Figure 8.10 Information model for the Proxy

The object type DO has a unique interface which consists of only one operation, namely **Invoke**, which is similar to the Invoke operation previously shown in Figure 8.6.

For an arbitrary object to interact with an object residing in a remote domain, a proxy representing the remote object must be present in the local domain. There are now two problems to solve. First, when an object invokes an operation on a remote object, the invocation must be redirected toward the proxy. Both a naming conversion and an object type conversion are necessary because the client generates an operation request using the reference of the remote server which implicitly refers to the object type of the server object. Second, the operation request must be translated and marshalled into the Invoke operation which is the only operation of the proxy.

To preserve transparency, these two problems must and can only be solved in the Engineering viewpoint and not in the Computational viewpoint. Correspondence must, however, exist between the two viewpoints.

Let us reconsider the example of a computational object CO1 invoking an operation OpX on a remote computational object CO2. When requesting an operation OpX on CO2, CO1 has to specify the Computational Interface Identifier (CII, also called Computational Interface Reference in TINA [TIN95a] of the interface I2 of CO2, (say CO2.I2) and the operation name OpX plus all the necessary operation arguments.

Let us now move to the engineering viewpoint. Suppose that CO1 and CO2 are mapped to the asic engineering objects BEO1 and BEO2, respectivley. Depending on the application designer's decision, the operation request can be built in a static way before compilation or dynamically at run-time. This will lead to different types of stubs that should be linked and compiled together with the programming code of BEO1. In the first case, the generated stub is specific to the interface CO2.I2 while in the second case the stub is generic and can be used by any object and in any binding. Dynamic Invocation Interface is an example of the second case. However, in both cases, the engineering operation request must contain the same information as the corresponding computational operation request, i.e the Computational Interface identifier, the operation name and parameters.

At run-time, when an operation request is issued, a channel must be established to interconnect the involved basic engineering objects. The necessary information to create a channel is contained in an Engineering Interface Reference (EIR) [ITUc]. Contrary to the CII, the EIR is both time and space dependent. The relationship between CII and EIR is shown in Figure 8.11.

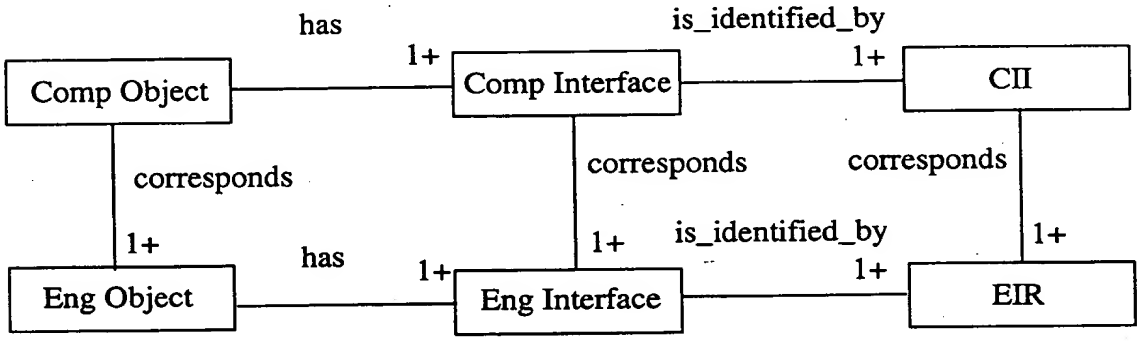


Figure 8.11 Relationship between objects, interfaces and identifiers in the computational and engineering viewpoints.

The DPE or more specifically the nucleus of the DPE node where the call is issued,, will be implicated to map the incoming CII to an EIR. This can be done in different ways, for instance, as an exception call in the stub code. The EIR will be used to establish the channel.

In our example, no EIR will be found for BEO2 since it is residing in another domain. Here, the DPE can be equipped with a redirection function which maps the incoming CII, i.e the CII of BEO2, to the EIR of the proxy PBEO2 instead of the EIR of BEO2. The mapping is forced in the sense that the proxy PBEO2 does not support the interface CO2.I2 and the operation OpX. A channel can now be establish between BEO1 and PBEO2. When the operation request arrives at the end of the channel, a conversion is necessary before the delivery to PBEO2. The server stub of PBEO2, instead of performing the usual unmarshalling of the request, has to convert it and pack it into the operation `Invoke` of the PBEO2 which may look like this:

```
Invoke (CO2.I2, Context, OpX, Arguments, Result)
```

The OMG's Dynamic Skeleton Interface allows indeed the specification of such flexible interface for a server object and the generation of stub from the interface specification. By this technique, the interface and operation name specified by the client object does not have to match with the interface and operations defined at the server object and can be compiled separately. In CORBA revision 2.0, there is also defined a redirection function as described above which supports the DSI.

The operation `Invoke` of PBEO2 can now be designed to contain a call to the operation `Invoke` of the user agent of the GMS. By this way, the operation request is passed to the GMS and conveyed from one domain to the another domain across the boundary. When receiving the request, the user agent in the other domain will invoke the operation OpX() on BEO2. The result can then be conveyed the same way back, first to the PBEO2 and then finally to BEO1. In Figure 8.12, the additional redirection function of the DPE and the special server stub of PBEO2 are emphasized by using darker colour than on other DPE's objects.

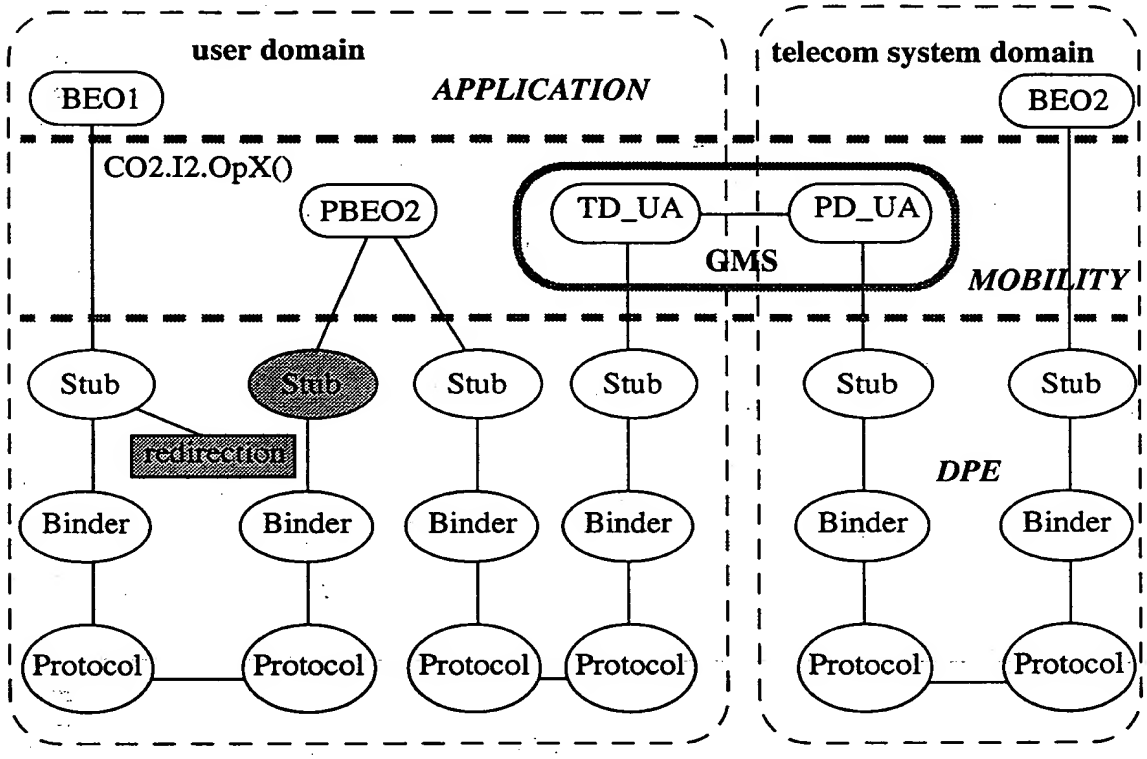


Figure 8.12 The redirection of operation request to the proxy

By using the proxy objects as described above, greater flexibility is achieved. The application designer does not have to perform the domain allocation of his objects, i.e to decide which objects should be allocated to which domain in the implementation phase. Only the usual grouping of objects into clusters, capsules and nodes is required in the transition from the computational viewpoint to the engineering viewpoint. Necessary stubs and skeletons can then be generated automatically from the computational specifications by the development tools. The application designer can proceed with the implementation of application specific programming code. The implementation of the application is completed with the linking and compilation of the application code and the generated stubs.

The definition and creation of proxy objects can be postponed until the configuration and deployment of the application. Depending upon the chosen configuration, the necessary proxy objects will be identified and defined. The proxies can be created at the initialization of the application and remain alive during the whole life time of the application. They can also be created dynamically according to the interactions and terminate upon the completion of the interaction.

As stated in Chapter 6, any object wanting to interact with objects in a different domain must be registered in the GMS. The registration of the necessary objects for an application can be done at the initialisation by an object which is customized for the configuration. The definition and creation of a proxy can be conveniently done together with the registration of the object it is representing.

In order to recapitulate what is achieved in this alternative, let us return to the same example considered in the two previous alternatives with for computational objects CO1, CO2, CO3 and CO4. In this alternative, it is not necessary to do the domain grouping in the transition from the computational viewpoint to the engineering viewpoint. Only the grouping of objects into clusters, capsules and nodes is performed. CO1 and CO2 belong to the cluster 1, CO3 to cluster 2, CO4 to cluster 3. The clusters 1, 2 and 3 are respectively within capsule A, capsule B and Capsule C. From this computational model, an Engineering model is derived and the necessary stubs are generated. Between CO2 and CO3, there is a need for a channel. The same applies to CO3 and CO4. Note that in this model all objects, clusters, capsules and nodes are considered as if they are all located within the same domain. The application designer can now add application specific programming code and execute the linking and compilation. The design and implementation of the application is then completed.

At installation, the plant engineer configures the application according to the request of the customer. Based on the configuration, he can derive the registration of objects to the GMS and the creation of necessary proxies.

This alternative yields very high level of flexibility. The inconveniences are the requirement for a new redirection function on the DPE and the possibility to generate special stub for the dynamic objects. These two inconveniences are however insignificant since DPE's such as CORBA revision 2.0 implementations already include these features.

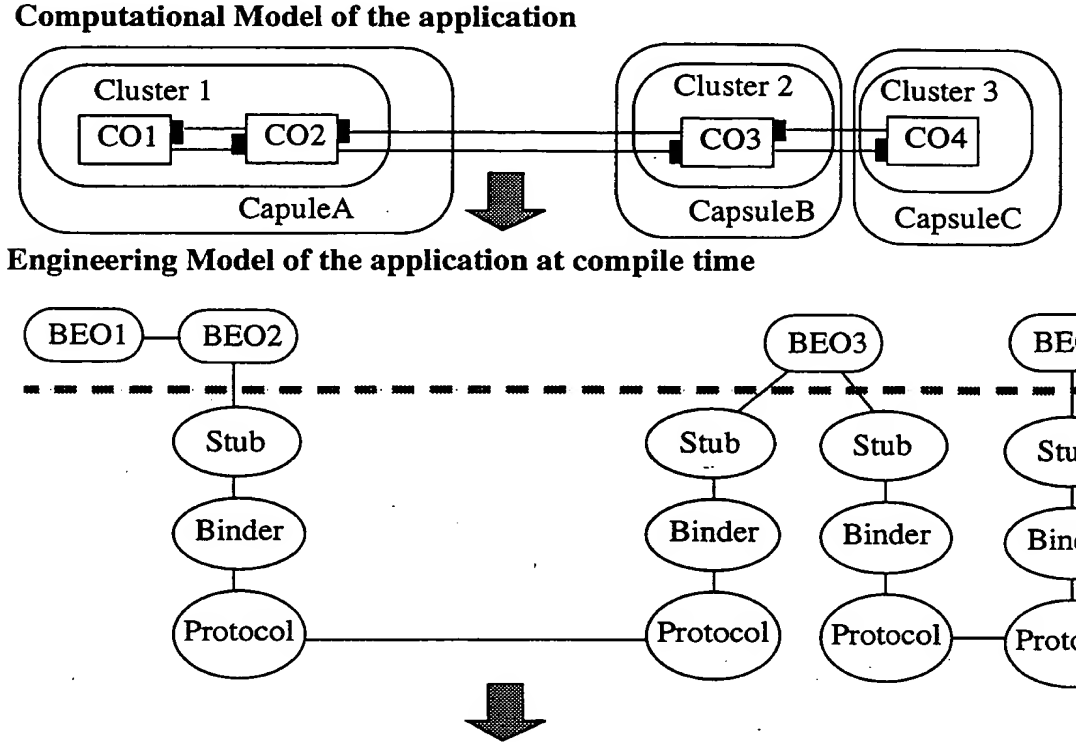


Figure 8.13 The Proxy alternative

8.6 CONCLUSION

In this chapter we have presented and discussed the advantages and disadvantages of three alternatives to integrate the Generic Mobility System (GMS) into the system: The in-lIne alternative, the broker alternative and the proxy alternative. The proxy alternative is chosen because it yields the highest level of transparency and also the highest level of flexibility.

9

Application Management

9.1 INTRODUCTION

Until now we have only study the functions and mechanisms necessary to support mobility at object communication level, i.e how objects belonging to a user domain interact with objects in the telecom system domain while the user domain and its contained objects and the underlying terminal domain are moving. We have not yet considered the application as an entire unit which consists of a set of interacting objects. There are several issues which need to be studied, such as how an application is made available to a user? What consequences has user mobility for the applications? Do the applications have to be aware of the user mobility? Do they need to have built-in functionality to cope with mobility? We shall attempt to clarify all these questions in this chapter.

9.2 SOME CLARIFICATIONS

9.2.1 Distinction between application, service and session

The following definitions are used in this thesis.

By **application** is meant a distributed system consisting of a set of objects that interact and make use of the functions supported by the DPE to perform certain functions. An application is designed and implemented by an application designer.

When an application is configured and installed in the telecom system domain or Service Provider system, it becomes a **service** to be offered to the user. Several applications can be combined and offered as one service to the user.

A service is brought to the user through a **session**. The session concept is adopted from the TINA service architecture but interpreted with some differences[TIN95k], [TIN96f], [TIN95j], [TIN96d], [TIN95c]. We will point out the differences in our discussion when it seems natural.

9.2.2 Session concept

TINA Service Architecture defines the session concept as follows:

"A session comprises a collection of objects and their relationships or associations, whose goal is the satisfaction of the purpose of a service. The session is associated with the allocation of resources that are necessary to execute the service and the association of parties using the service. In TINA-C a service is instantiated according to the session concept and it is provided as a session."

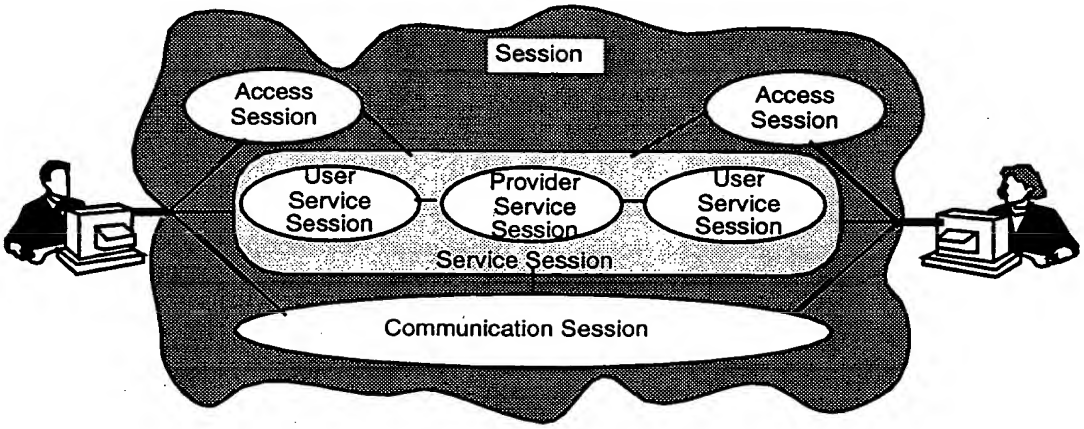


Figure 9.1 The Session Concept

We interpret a session simply as a period of activity that a user has with the telecom system domain or service provider domain.

There are several types of sessions:

An **access session** is a period of activity that is concerned with allowing the user to access the telecom system domain. Identification, authentication, access control and registration of the user and the interactting domain are the activities carried out during an access session.

A **service session** is a period of activity that is concerned with the provision of a service to a user. A corresponding application is then activated and run during the service session.

A service session can be decomposed further into **User Service Session** and **Provider Service Session**. A User Service Session represents information related to a user's customized view of a service. A Provider Service Session represents information related to service capabilities shared among users.

A **communication session** represents a network technology independent view of communication resources (e.g. stream bindings that are end-to-end connectivity). A communication session may include multiple multi-media multi-point connections.

The relations among Access, Service and Communication Sessions as shown in Figure 9.2

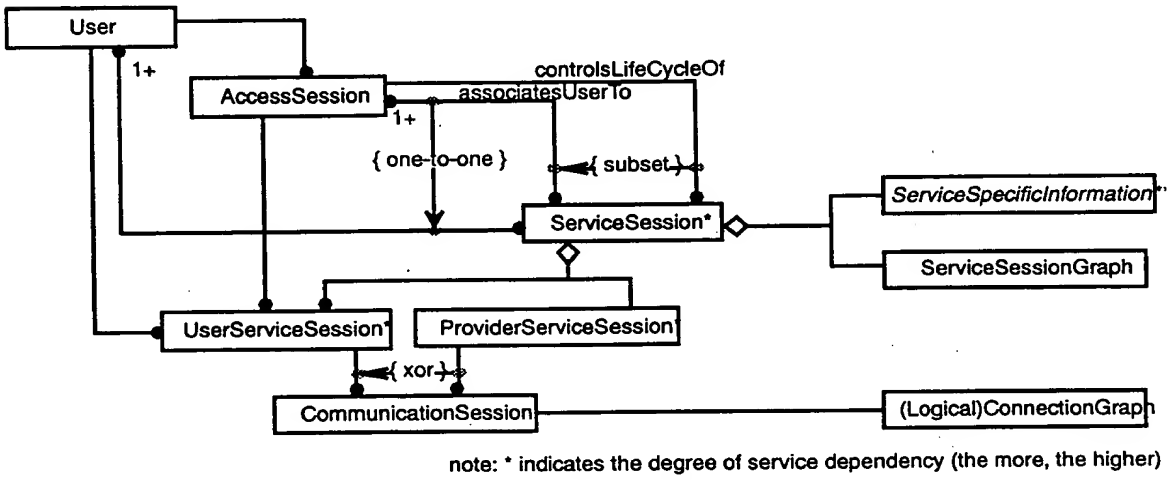


Figure 9.2 A Partial Sketch of the Session Information Model

According to Figure 9.2, a `User` may be associated with zero or more **AccessSession** and zero or more **ServiceSessions**. The **AccessSession** includes information necessary to initiate new **ServiceSessions** and to receive invitations to join existing **ServiceSessions**. Each **ServiceSession** may be associated with one or more users (one of them will be allowed to control the life-cycle (suspension, resumption, deletion) of the **ServiceSession**). One **ServiceSession** may have, at a given time during its lifetime, zero or more **CommunicationSessions**, which may have zero or more point-to-point or point-to-multipoint connections that may be based on different network technologies. For a given **CommunicationSession**, there is one **ServiceSession** controlling the **CommunicationSession**.

In this thesis, we choose to expand the relation between **User** and **AccessSession** by allowing one user to be associated with zero or more access sessions. The TINA service architecture [TIN95j] only allows one.

Dependencies among session types exist and satisfy the following constraints:

- An access session exists only if a user has the rights to access a service.
- A service session can be in the active state only when an access session exists. The service session may be in the suspended state also when no access session exists. There is here a difference with TINA-C which states that a ser-

vice session can exist only if an access session exists. The reason for this difference is explained in Paragraph 9.6.2

- A communication session can exist only if a service session exists.

The notion of session is generic and the TINA Service Architecture has also defined a GenericSession object [TINA:servic_archi]. A property of the GenericSession is its prolongation in time. During its lifetime, a GenericSession changes between different states: initial, active, suspended, limited, resuming, completion. A session is owned by a user.

In order to support session mobility, unique identifiers are to be assigned to access sessions and service sessions (Provider Service sessions and User Service sessions). The PD_UA will be the object responsible for the management of all the user's sessions. We shall see later that the PD_UA will have the capability to suspend, resume or delete a session.

Before proceeding to the study of the access session and service session, it is necessary and useful to build an information model for the user subscription.

9.3 INFORMATION MODEL FOR THE USER SUBSCRIPTION

In order to access services a user must have a subscription with some providers in the telecom system domain. An information model for the user subscription is shown in Figure 9.3. This is a generic model applicable to most telecommunications applications. It is worth mentioning that this information model is similar to the one of Universal Personal Telecommunication (UPT) presented in [Do96d]. This is not surprising since the UPT concept, which primarily applies for telephony can easily be extended to incorporate all types of services. We have used qualifiers to make the model easier to understand. The objects in the model are

The **Service_Provider** has overall responsibility for service operation and on database management. The **Service_Provider** will commercially manage the services. The **Service_Provider** offers **Services** to the **Users**

The **Subscriber** is a person or legal entity with a contractual relationship with a **Service_Provider**, where the subscription may be done on behalf of one or more Users. The subscriber is also responsible for the payment of charges due to that service provider. Each **Subscriber** has the relationship **Subscription** with the **Service_Provider**. The subscription is made available to the **Users** by the relationship **User_Profile**.

The **User** is a person who has been authorized to use services subscribed to by a **Subscriber** through the relationship **User_Profile**. The User's access to Services is defined by the relationship **Access_Control**. The **Access_Control** is a security service which is treated in Chapter 7.

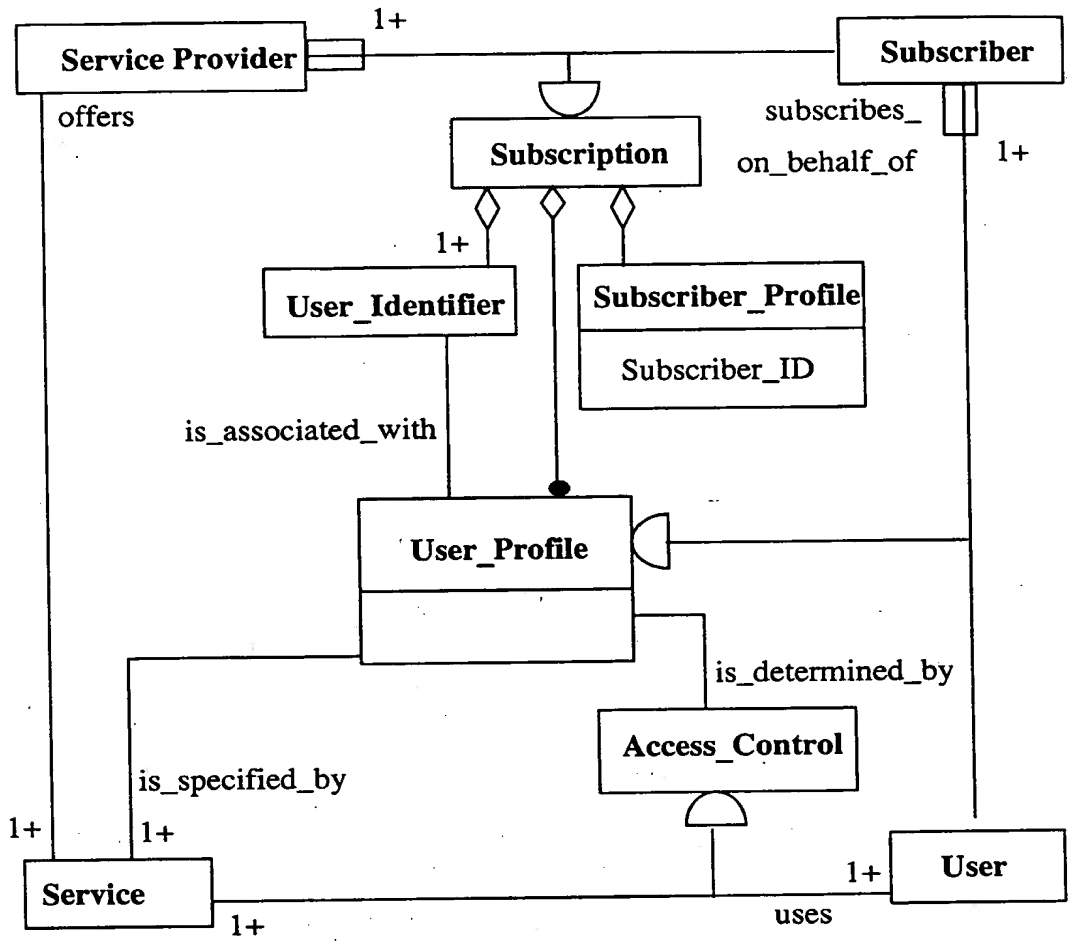


Figure 9.3 The information model for the user subscription

The Relationship type **Subscription** which is also an object, contains one or more **User_Identifiers** which are associated with one **User_Profile,** one **Subscriber_Profile** and zero or more **User_Profile.**

The **User_Identifier** uniquely and unambiguously identifies a **User.** The **Subscriber_Profile** is a personalized data record for each subscriber, detailing the services and listing his users, and contractual terms between the subscriber and the service provider. This data may be used for billing the subscriber.

The Relationship type **User_Profile** is a record containing all the information related to an each user in order to provide that user with services. Each **User_Profile** is associated with a single **User_Identifier**. The relation type **User_Profile** again is composed of five objects, **Service_Restriction, Routing_Info, Charging_Info, Security_Info** and **User_Application_Profile** as shown in the Figure 9.4.

Service_Restriction has attributes such as

- Roaming restriction
- Time restriction
- Credit limit
- Maximum number of terminal addresses for group registration for incoming applications
- Incoming screening
- Outgoing screening
- List of subscribed services

Routing_Info has attributes such as

- Forwarding activation status
- Registered terminal address for incoming applications
- A linked-registered terminal address
- Default terminal address for incoming applications
- Routing by applications originating area
- Routing by calling party identity
- Time-dependent routing
- Routing on "busy" condition
- Routing on "no answer" condition
- Default duration (or number of calls) for incoming applications registration

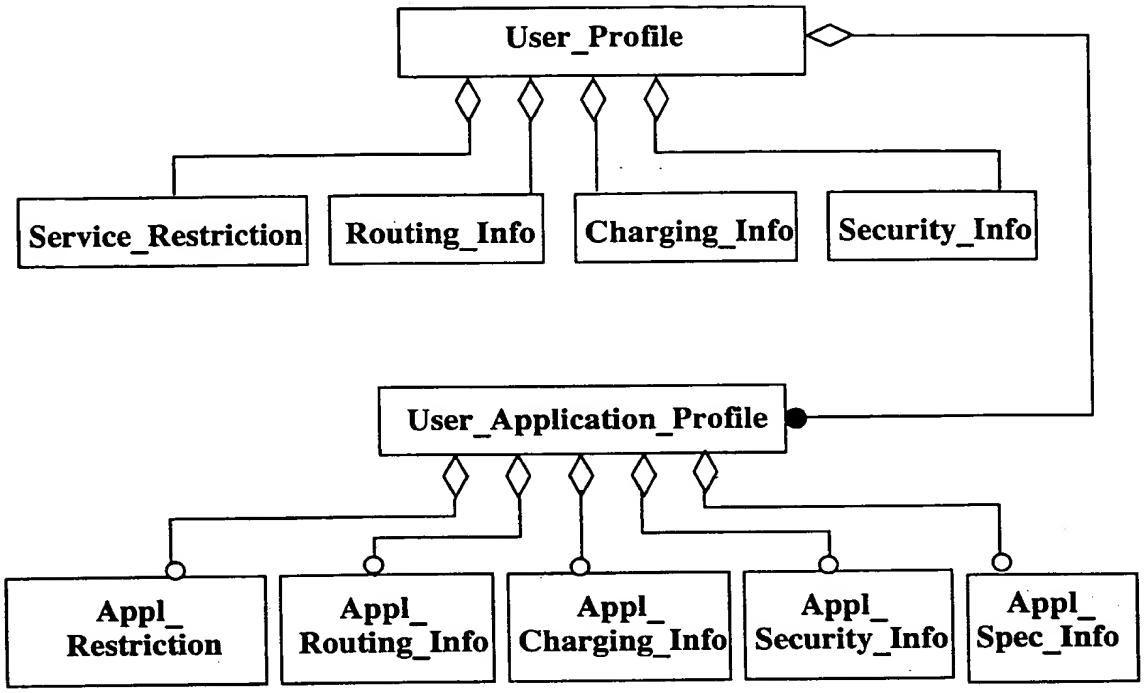


Figure 9.4 User Profile composition

Charging_Info has attributes such as

- Default charging reference location
- Charging option selected
- Temporary charging reference location
- Advice of charge activation status

Security_Info has attributes such as

- Authentication procedures subscribed
- Security options subscribed
- Type of authentication procedures activated
- Max number of failed authentication attempts
- Password

The **User_Profile** may contain zero or more **User_Application_Profiles**. The **User_Application_Profile** component is to enable customisation of an application. For each application (run in a service session), there may hence be assigned zero or one **User_Application_Profile**. The **User_Application_Profile** may contain zero or one **Application_Restriction**, **Application_Routing_Info**,

Application_Charging_Info, **Application_Security_Info** and **Application_Spec_Info.** It is therefore possible to specify the restrictions, routing, charging and security options for each application. The **Application_Spec_Info** is a component that contains application specific data. Greater flexibility is achieved in this way. An application must, however, not have its own **User_Application_Profile**. For applications that do not have their own profile the main **User_Profile** is applied at the initiation of the application.

The user can subscribe to a service which permits him to interrogate and modify some of the attributes of his **User_Profile**. His access rights are linked to and used by the access control procedure (see Chapter 7).

9.4 ACCESS SESSION

The purposes of an access session are to separate the access capability from the service core capability and to enable a user to access a variety of services. An access session is started when the user logs in at the telecom system domain in order to use services or to receive an incoming service, e.g an incoming call. If the access is successful, the access session will exist until the user terminates all his activities and logs out. Otherwise, the access session will be concluded immediately. An access session comprises multiple procedures such as identification, authentication, access control and user registration. Note that there may exist several access session simultaneously for a user, e.g one for accessing the network, one for accessing an Internet server and one for accessing content. These accesses may succeed or fail independently of each other.

Although there are several computational objects involved in the access session, the most central is the `PD_UA` that represents the user. The `PD_UA` shall assume the responsibility to save and manage the **access session identity** (ASI) for the whole period in which the access session exists and to remove this identity when the access session terminates. An access session is related to a terminal and since a user may use several terminals at the same time, he may have multiple access sessions running at different terminals.

It is worth noting that the access session covers more than the user registration procedure (see Paragraph 6.4.3). When a user is registered, it means only that a `TD_UA` on a specific terminal is defined and assigned to him. This does not necessarily mean that he has the rights to access the telecom system domain in order to use services. He may have to identify himself, be authenticated and be subject to the access control.

The object `User_Registration` which saves data related to the user mobility, is extended to contain also information about sessions. As shown in Figure 9.5, a new column called `SessionInfo` is added to the model presented in Figure 7.9. Each row of this column contains an `Access_Session_Info` and a pointer to `Service_Session_Table`. The `Access_Session_Info` contains informa-

tion such as Access_Session_Identity (ASI) and Access_Session_State which can take the values Initial, Suspended, Active, Limited, Resuming, Completion. The Service_Session_Table contains information about all the service sessions belonging to the access session. Such information can be Service_Session_Identity, Service_Session_Type (user or provider), Service_Session_State which take the values be Initial, Suspended, Active, Limited, Resuming, Completion.

The **User_Registration** object must be equipped with the following additional operations in order to manage the new information content:

AccessReg(in ASI,in ASState,in TAI) to register an access session.

AccessDereg(in ASI,in TAI) to deregister an access session.

AccessChange(in ASI,in ASState,in TAI) to change the state of an access session.

ASI is the access session identifier, ASState is the state of the access session and TAI is the TA identifier.

As stated earlier, the **PD_UA** will also be responsible for the management of all the user's Service sessions. The **PD_UA** may suspend, resume or delete a service session. The operations **SSAdd()**, **SSRemove()**, **SSModify()** and **SSGet()** must be defined for **User_Registration** object in order to add, remove or modify a service session. The input parameters should be the ASI, the Service_Session_Id, the Service_Session_Type and the Service_Session_Type. The new **User_Registration** object is shown in Figure 9.5

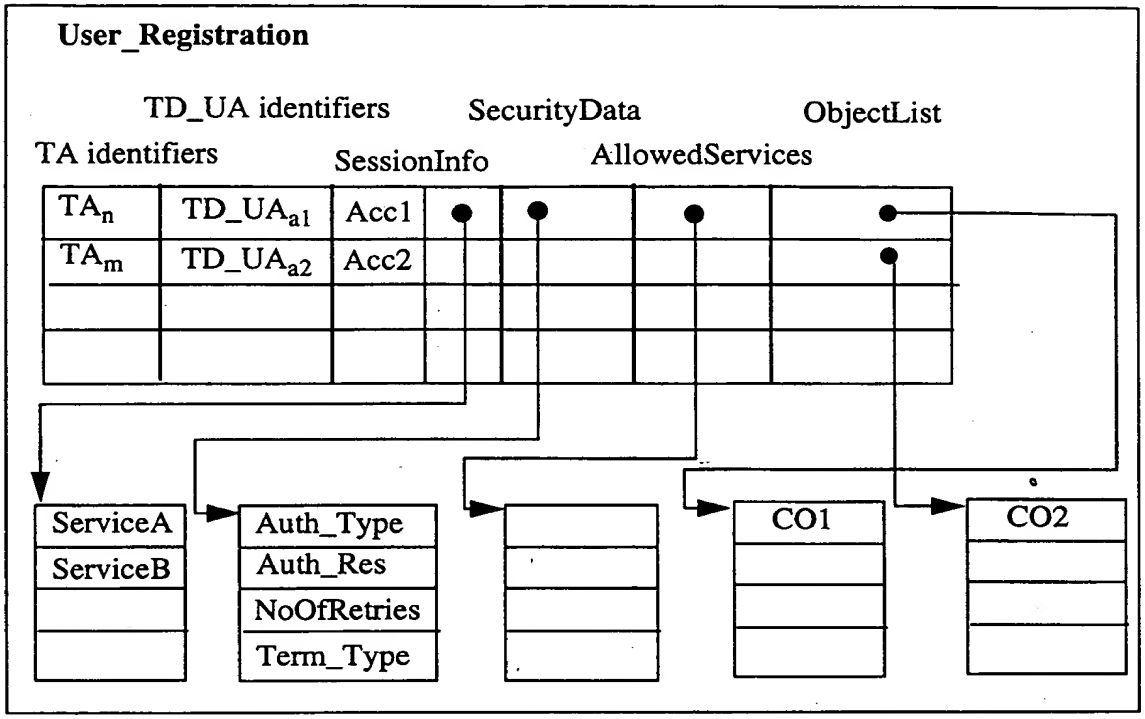


Figure 9.5 A User_registration object extended with the access session identity and service session information

The computational objects involved in the access session are shown in Figure 9.6. The activities carried by the access session are listed chronologically as follows:

1. When $\texttt{User}_a$ wants to use terminal_m he/she can issue a request `New_User` (by hitting some key or using a mouse to click, etc.). The `UI` (`User_Interface` object defined in Paragraph 6.4.3.4) creates a new and "temporary" `TD_UA` object. By temporary, it is meant that this `TD_UA` is not yet associated to any user or specifically to any `PD_UA` which is yet known and recognisable for the telecom system domain. The association will be done after the local registration has been accomplished successfully. The `UI` will also connect the input/output channel to this `TD_UA`. The $\texttt{User}_a$ can now interact with the temporary `TD_UA`.

2. $\texttt{User}_a$ enter his User Identifier (or name) to the temporary `TD_UA`. From the User Identifier the temporary `TD_UA` deduces Computational Interface Identifier (CII) of the corresponding $\texttt{PD\_UA}_a$ and invokes the operation `LocalRegister()`. `TD_UA` can however not invoke it directly but needs to incorporate it in a call operation on the $\texttt{SPA}_N$ as follows: `Call(`$\texttt{PD\_UA}_a$`.LocalRegister())`. The invocation is conveyed through the `TAP` and `NAP` and arrives finally at $\texttt{TA}_m$.

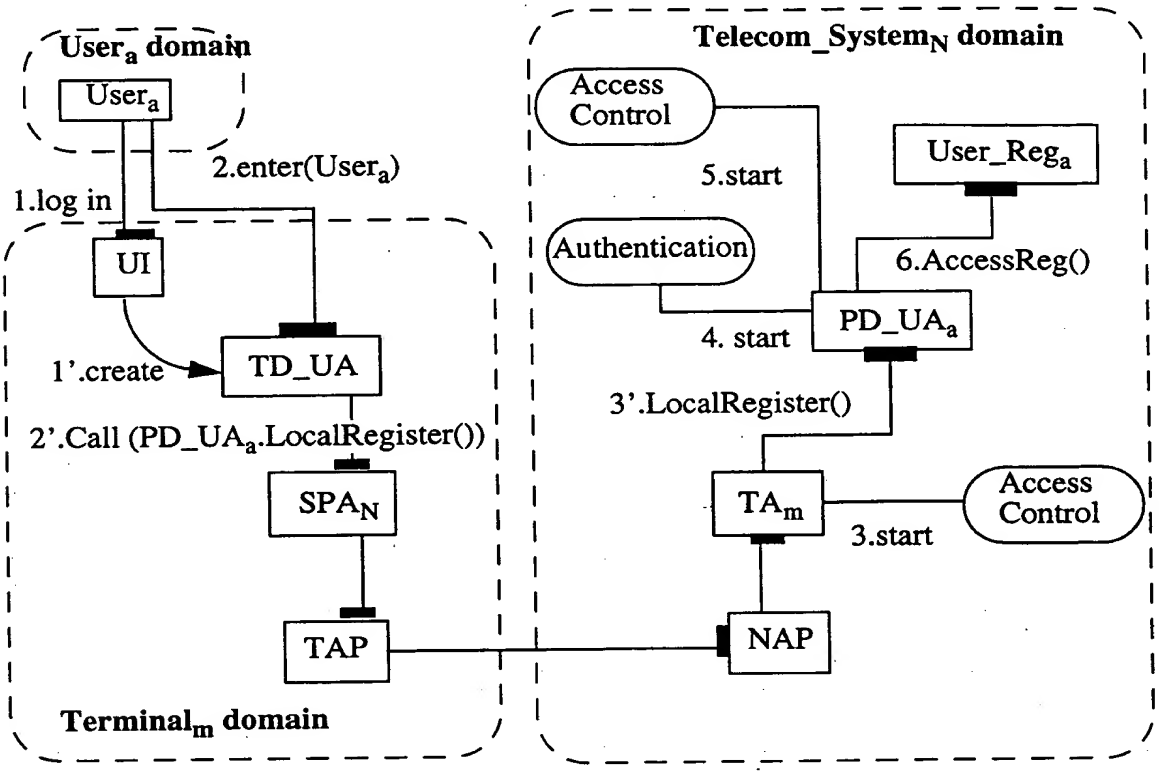


Figure 9.6 The Computational objects involved in the access session

3. TA_m checks with the object `Terminal_Data` and finds that $User_a$ has not been cleared for use of the terminal. TA_m starts therefore the access control procedure (see Chapter 7). If the access control fails, the access session terminates here with a denial message to the $User_a$. If it is successful, TA_m will then invoke `LocalRegister()` on the PD_UA_a.

4. PD_UA_a will start the authentication procedure to make sure that the user really is the one he claims to be. The authentication procedure is described in Chapter 7 and will not be repeated here.

5. PD_UA_a starts the access control procedure (see Chapter 7) toward the telecom system domain. If this access control fails, the access session terminates here with a denial message to the $User_a$. If it is successful, the PD_UA_a proceeds with a user local registration (See Chapter 6).

6. PD_UA_a invokes `AccessReg(ASI,Acitve,`TA_m`)` on $User\_Registration_a$ to register the access session identity. The access session will then exist until the user logs out.

7. PD_UA_a can now start the first service session. The first service session can be "empty", i.e do nothing but waiting for a request from the user. It can be a menu with all the available services that the user can just click to choose. It can also be a voice menu, using a user-defined language and telling the user what to do. The application running in the first service session will be referred to as the Main application. In principle, there is no difference between this first service/application and the other ones (for more details, see Paragraph 9.5.2.3). We shall see later that it will be possible for the user to customise any service or application that he is allowed to use.

The computational model for the access session presented above differs considerably from the one suggested by the TINA Service Architecture [TIN96f]. In the TINA Service Architecture, the list of services is presented to the user by the **User Agent** which acquired the list from the **Subscription Agent**. The user makes the choice of the service with the **User Agent** using the operation `Request_Service()`. This solution requires that the **User_Agent** must be equipped with several different input/output capabilities in order to present the list of services to the user if the user can use terminals with different capabilities. Another remark concerns the list of services. As explained in Chapter 7, the set of services a user can use at a terminal may be much smaller than the list of subscribed services. This is due to the limitations in the terminal capability, the access control and/or the authentication mechanism used.

9.5 SERVICE SESSION

As mentioned before, a service session carries one or more applications. Let us start by studying in details an application.

9.5.1 application structure

In order to offer functions to a user, an application needs to be able to communicate with the user. An input/output object is therefore required. This I/O object can be part of the application or can belong to another application which can interact with objects of the former application. A computational model of an arbitrary application is as follows:

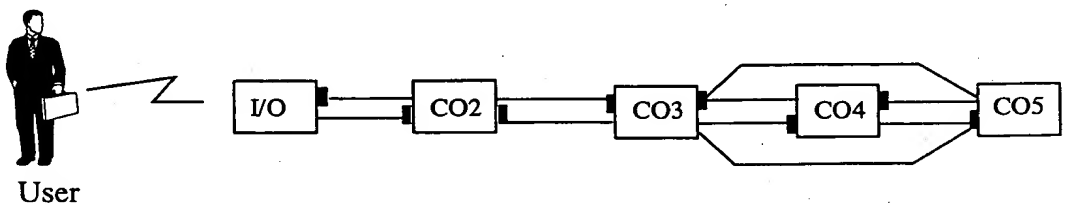
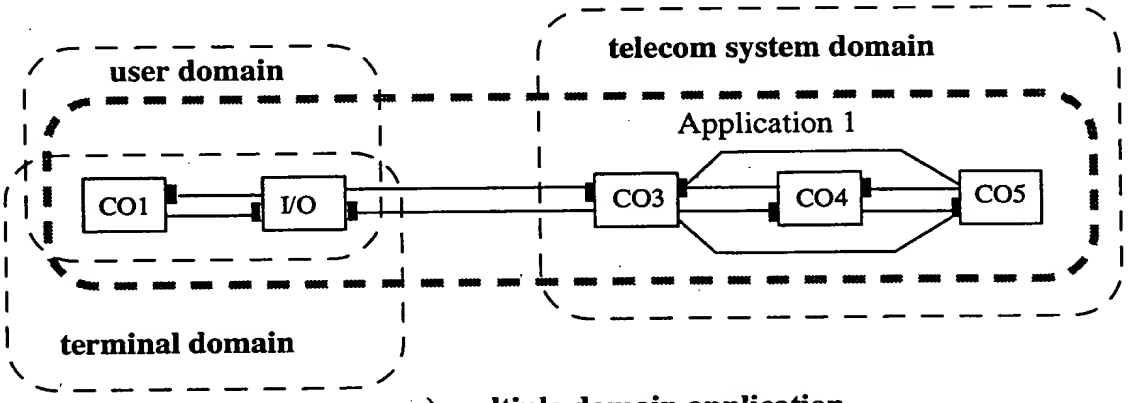
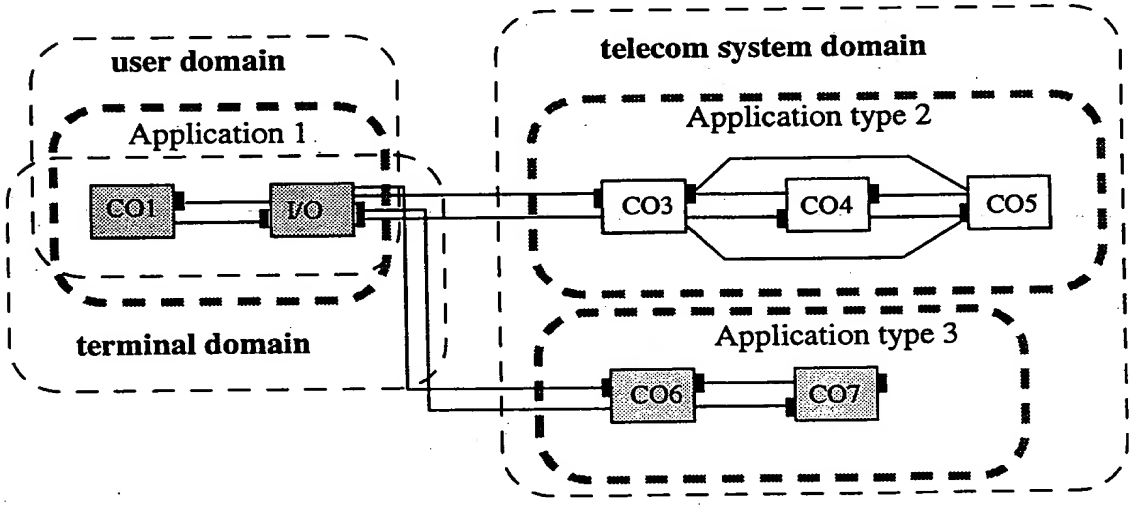


Figure 9.7 Computational model of an arbitrary application

As discussed earlier, a system consists of three domains: the user domain, the terminal domain and the telecom system domain. The terminal domain is, of course involved indirectly since all the objects of the user domain are residing or communicating with the telecom system domain through the terminal domain. The I/O object belongs naturally to the user domain (and run of course on the terminal domain) and not to the telecom system domain. Taking this into account, applications can be classified into two types, **Multiple Domain** applications and **Single Domain** applications, as shown in Figure 9.8.



a) multiple domain application



b) single domain application

Figure 9.8 Two types of application

The Multiple Domain type application has computational objects in both the user domain and the telecom system domain. One of the objects on the user domain is the I/O object. At run-time, a Multiple Domain application can only work if two conditions are satisfied:

- First, objects belonging the user domain must be created on the correct terminal domain and must be able to function given the capability of the terminal domain, i.e the resources (memory, graphic interface, etc.) required by the objects must correspond to those offered by the terminal. In the example shown in Figure 9.8, CO1 and I/O must be created on the correct terminal and must be able to run on it. The interfaces of objects on both sides are mutually known since all objects are designed together.
- Second, the Computational Interface Identifiers (CII) of objects having inter-domain interactions must be mutually passed to their counterpart. Recall that a CII identifies uniquely a computational object and its interface [ITUc]. In our example, the I/O object must have the CII of CO3 and, reciprocally, CO3 must have the CII of I/O object.

The Single Domain type has all its object on one domain, i.e either on the user domain or on the telecom system domain. However, an application which is entirely on the user domain (or more precisely in the terminal domain) is nothing else but a local application and falls beyond the scope of our study. In the case in which the application is in the telecom system domain its objects must be able to interact with an I/O object belonging to another application in the user domain. The second application may be designed independently of the first or by the same application designer or team. The I/O object of the second application can be created independently of the first applications. As shown in Figure 9.8, the I/O object in one application can interact with objects of more than one applications. An example of such application is a terminal that can serve several applications at the same time . At run-time, such a Single Domain application can only work if two conditions are satisfied:

- First, objects having inter-domain interactions must know how to communicate with the I/O object, i.e the interfaces of both sides are mutually known to each other.
- Second, the CIIs of both sides must also be mutually known in order to be used in operation invocations.

Let us consider the first condition for both types of applications. For Multiple Domain application, when the user is moving and using different types of terminals, different type of I/O objects must be used and created at the correct terminal. For Single Domain type, the problems are which I/O object to communicate with and how to communicate correctly with it since there may be many of them with different interfaces. These two conditions are actually different sides of the same thing. In fact, both the I/O object type and interface reference can be derived from the terminal identity.

They can hence be partly satisfied at the design and implementation phase and partly at run time as follows:

For different types of terminal with different capabilities, different types of I/O objects are required. In order to enable applications to function at terminals of different types, an `Adapter` object is usually introduced. All interactions with the I/O object shall always go through the Adapter object. There may be many types of Adapter object having different interfaces with different types of I/O objects but they all have the same interface toward all other objects. In a fixed system, the types of I/O object and Adapter object to be used for a terminal are defined at configuration and installation of the application. When the terminal type and its capability are known, the application can use the appropriate Adapter. Such application is shown in Figure 9.9.

When the user is moving and using different terminals, the requirements are now:

- To create the correct type of Adapter which fits the terminal type.
- To pass the CII of the Adapter to the I/O object and vice versa. Who owns and creates the I/O object is not important anymore.
- To pass the CII of the Adapter object to the appropriate application objects and vice versa.

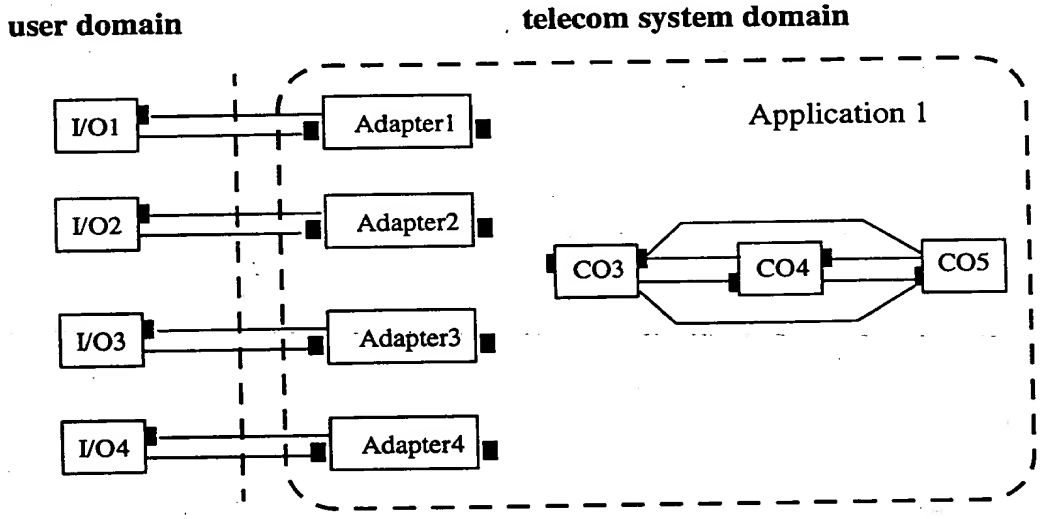


Figure 9.9 An application with several I/O possibilities

These conditions are "run-time" conditions and cannot be fulfilled at the design and implementation of the application. Since applications are brought to the user as a service through a session, these conditions can only be fulfilled by the session concept.

Based on the definition and decomposition of the service session described in Paragraph 9.2.2, the mapping of an application to the service session is as shown in Fig-

ure 9.10. The service session is decomposed into three session components: Provider Service Session, Provider Domain User Service Session (PD USM), Terminal Domain User Service Session (TD USM). The Provider Service Session groups all the objects which are generic for the application, such as CO3, CO4, CO5 shown in Figure 9.10. The PD USM groups the objects which can be customized by the user and are dependent on the current terminal used by the user, such as the Adapter object. The TD USM groups the objects running on the terminal domain, such as the I/O object.

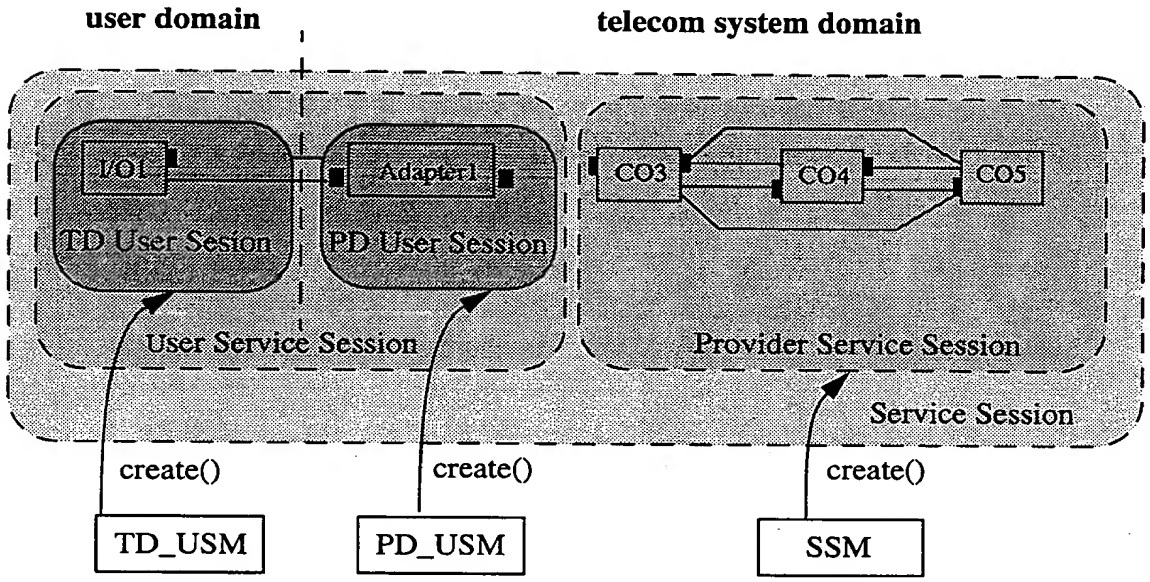


Figure 9.10 The mapping of an application to the Service session

To manage the three service sessions, i.e create, suspend, resume, and terminate them, three computational objects are introduced:

- Service Session Manager (**SSM**) to manage the Provider Service Session.
- Provider Domain User Service Session Manager (**PD_USM**) to manage the Provider Domain User Service Session.
- Terminal Domain User Service Session Manager (**TD_USM**) to manage the Terminal Domain User Service Session.

We will see later that these objects play a central role in the initiation of the service session and also in the support of session mobility.

Depending on the initiator of the service session different conditions and data are required in order to establish a service session. It is therefore useful to classify applications into different types based on the initiator:

- **"Outgoing"** applications are run on a service session initiated by the user.
- **"Incoming"** applications are run on a service session initiated by some other party than the user we are considering.
- **"Outgoing Incoming"** applications are the combination of the two previous types. Communications between one or several users are typical examples of this types of applications. Broadcast can be realised as an outgoing application combined with multiple individual incoming applications.

Note that when we say that a service session is initiated by the user, the initiation may be done through other service session which are initiated by the user.

We shall study in details successively the two first types of applications in the following paragraphs. The third type is then implicitly explained since it is the combination of the first two.

9.5.2 Outgoing applications

9.5.2.1 Definition

An outgoing application is initiated directly or indirectly by the user himself. The term outgoing is adopted from Universal Personal Telecommunication (UPT) vocabulary [Eri94] [ETS93] [ITU94] and used for outgoing calls. An outgoing application may be a traditional telecommunications application such as voice phone, videophone, etc. or a computing application such as word processor, database, spreadsheet, etc.

9.5.2.2 Information viewpoint

All the information necessary for the initiation of an outgoing application is described by the information model shown in Figure 9.11. One **Outgoing_Application** can be initiated at one or more **Current_Terminals** by different users or by the same user. Similarly, one or more **Outgoing_Application** can be initiated at one **Current_Terminal**. The relation between **Current_Terminal** and **Outgoing_Application** is called **Initiation_Restriction** and expresses the usage restriction of the outgoing application at the current terminal. This is determined by the **Application_Restriction, Terminal_Capability** and **Usage_Restriction** information objects. The **Usage_Restriction** is the relation between a **Current_Terminal** and a **Third_Party** owning the terminal.

The **Usage_Restriction** is to allow the **Third_Party** to limit the use of his terminal and is used in the access control procedure for the use of the current terminal described in Chapter 7. If the **Current_Terminal** is determined then the **Usage_Restriction** can also be found.

Once the **Current_Terminal** is determined, then the corresponding **Terminal_Capability** can be deduced.

A **User**, qualified by a **User_Identifier** has one or zero **User_Application_Profile** which contains user-defined data for the application. From the **User_Application_Profile** the component **Application_Restriction** can be found.

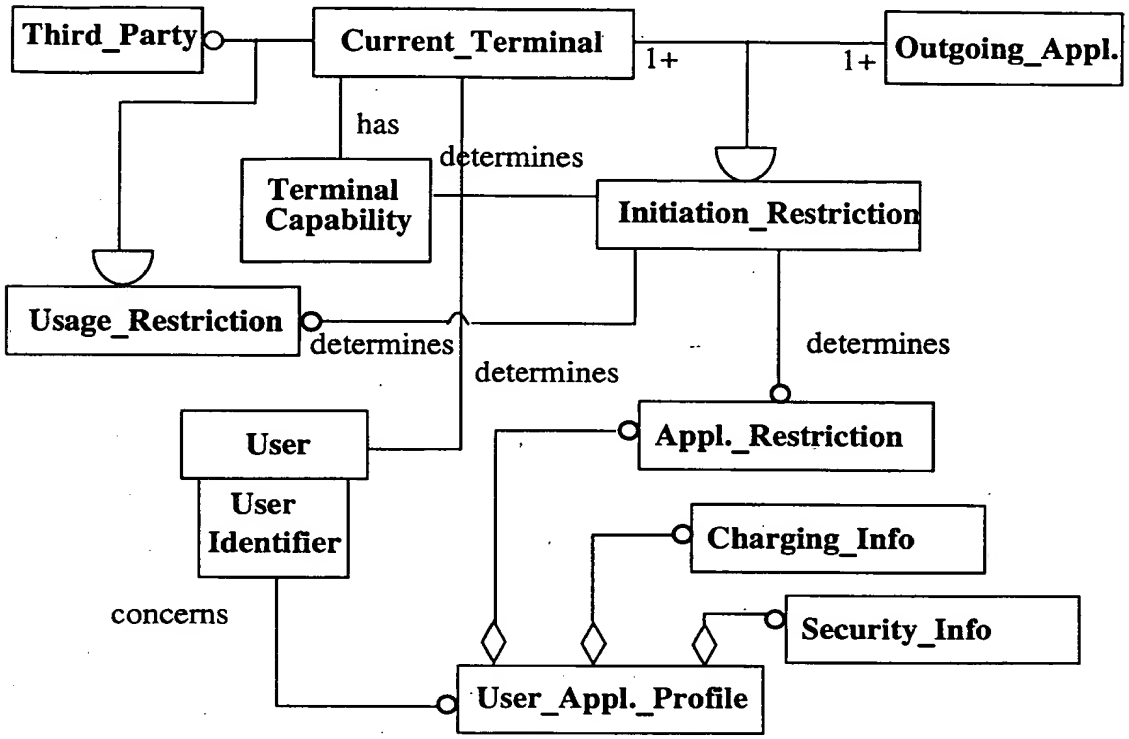


Figure 9.11 An information model for an outgoing application

All the three conditions necessary to establish the value of the **Initiation_Restriction** can always be determined and the initiation of the outgoing application can always be started. It may, however, be stopped by the access control procedure due to the information stored in the **Usage_Restriction**, the **Terminal_Capability** and the **Application_Restriction** information objects.

We will see later that in the case of incoming applications the conditions necessary to establish the value of the **Initiation_Restriction** are not always determined and the initiation is hence not always possible.

We shall now study two types of outgoing applications: the Main application and generic outgoing application.

9.5.2.3 Main application

The Main application is run in the first Service session right after a successful access session has been established. The Main application can be "empty", i.e does nothing but just waiting for a request from the user. It can display a menu with all the available services that the user can choose. It can also be a voice menu using a user-defined language and telling the user what to do. The Main application can also assume the responsibility to suspend or resume service sessions as requested by the user. The Main application may also be equipped with functionality enabling the user to make registration of multiple incoming applications. The registration of incoming application is treated in Paragraph 9.5.3.3.

There are, of course, several different ways to build a Main application, but for simplicity sake and without loss of generality, let us suppose as shown in Figure 9.12 that the Main Application consists of one I/O object, one object called Menu and one Adapter object. The type of Adapter object must correspond to the type of the I/O object used. The type of I/O object depends on the type of the terminal in use.

The prerequisite of the initiation of the first service session supporting the Main application is that an access session has already been successfully created. The successive steps of the initiation of the Main application are as follows (Figure 9.12):

1. $\mathbf{PD\_UA_a}$ invokes a `Get(User_Appl_Profile,Main)`on the $\mathbf{User\_Profile_a}$ in order to get the profile of the Main application which contains $User_a$ defined data for the Main Application if such a profile is defined. If such a profile is not defined, the main **User_Profile** will be used to set up the Main application.

1'. $\mathbf{PD\_UA_a}$ invokes a `Get(Capability)` operation on $\mathbf{TA_n}$ to get the capability of the $Terminal_n$.

2. $\mathbf{PD\_UA_a}$ invokes an operation `CreateService(Main,`$User_a$`,ASI, User_Appl_Profile,Term_profile)` on the **SF** (Service Factory) object. The **SF** object is an object that is responsible for creating the service session. We need this object to avoid burdening the **PD_UA** with object creation functions which are platform dependent. **SF** should have the capability to store all the created service sessions and their states. In the telecom system domain, there may be several instances of the **SF**s. This is, however, a matter of system configuration and does not have any impact on the proposed design. In each terminal domain there must also be at least one instance of the **SF** to create the **TD_USM** (Terminal Domain User Service Session Manager).

The input parameter `Main` indicates the name of the application. The input parameter $User_a$ indicates the owner of the service session. The input parameter `ASI` (Access Session Identity) is the identity the access session controlling the service session. From the `ASI` the **SF** can deduce the terminal identity and also the CII (Computational Interface Identifier) of an **SF** residing in the current terminal domain, in our example $\mathbf{SF_n}$ in order to request the creation of the **TD_USM** object.

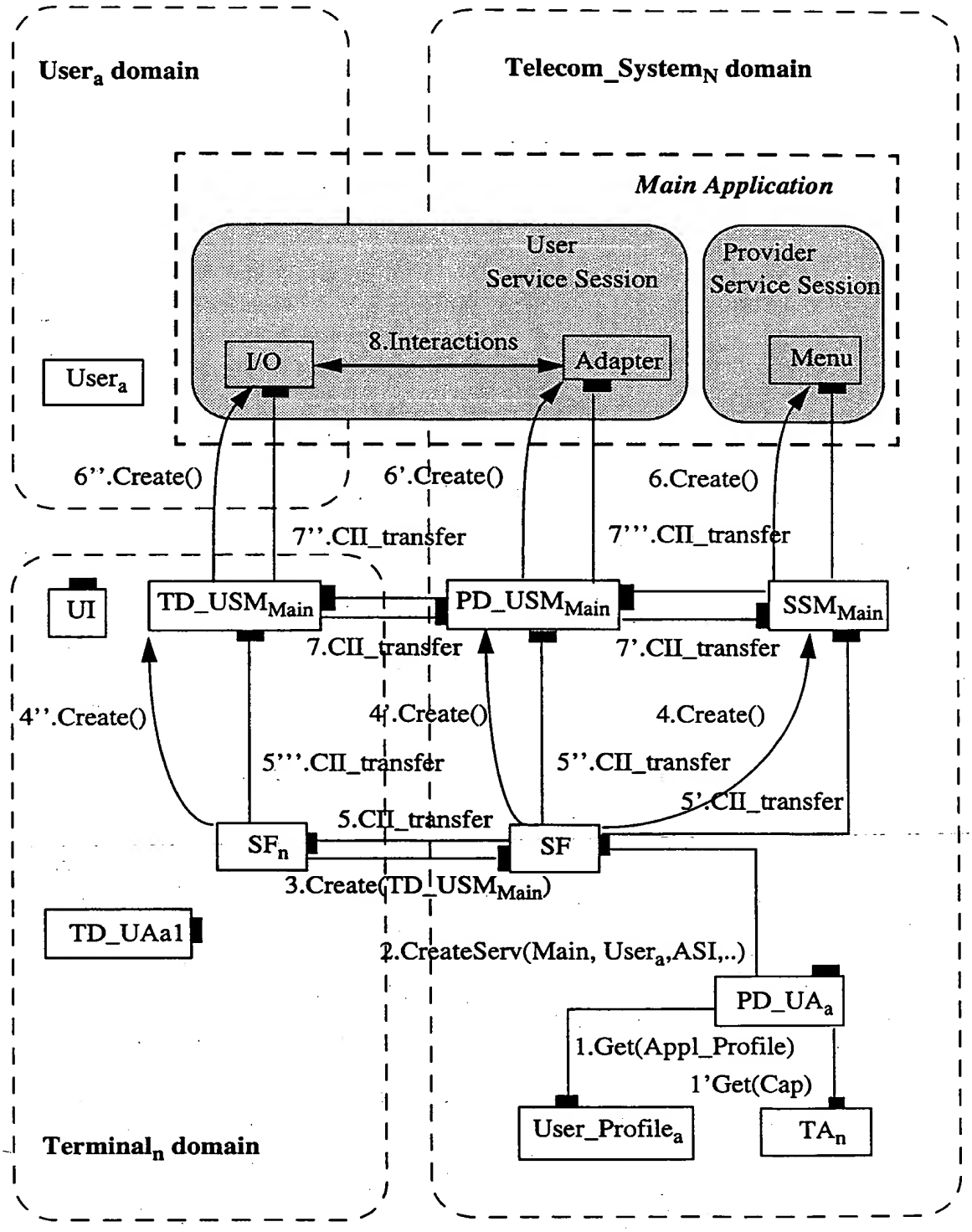


Figure 9.12 Initiation of the Main application

When the operation `CreateService()` is successfully completed, the `SSI` (Service Session Identity) and the CIIs of the SSM_{Main}, PD_USM_{Main} and TD_USM_{Main} will be returned to PD_UA_a. PD_UA_a saves this information in the $User\_Registration_a$ object. This information is used to support session mobility (see Paragraph 9.6.2)

3. `SF` deduces the CII of SF_n from the parameter `ASI` and invokes `Create(`TD_USM_{Main}`)`.

4, 4', 4". `SF` and SF_n create the session manager objects SSM_{Main}, PD_USM_{Main} and TD_USM_{Main}.

5, 5', 5", 5'''. `SF` and SF_n transfer the CIIs of the created session manager objects to each other, e.g the CII of TD_USM_{Main} is transferred to PD_USM_{Main} and vice versa. The created session manager objects (SSM_{Main}, PD_USM_{Main} and TD_USM_{Main}) are now able to communicate with one another.

6, 6', 6''. The session manager objects create the objects belonging to their session. In our example, TD_USM_{Main} creates the `I/O` object, PD_USM_{Main} object the `Adapter` object and the SSM_{Main} object the `Menu` object. It is worth to recall that during the object creation, objects in the user domain will be registered with the PD_UA_a (see Chapter 9) and proxy objects for objects having inter-domain communications will also be creates (see Chapter 8). We are not including these objects in the figure. The session manager objects are responsible for the initialization of all the application objects.

7, 7', 7'', 7'''. The session manager objects exchange and transfer the necessary CII of the created application objects to each other, e.g the CII of the I/O object is transferred to the `Adapter` object and vice versa.

8. From this step all the application objects are completely initialized and can operate properly. The initiation of the Main application is hence completed.

9.5.2.4 A generic outgoing application

Let us now study the initiation of a generic outgoing application called ***out application*** by $User_a$ using a $terminal_n$ as shown in Figure 9.13. More accurately, ***out application*** is started by another outgoing ***application XX*** which may be the main application. For simplicity, let us suppose that ***out application*** consists of one object `CO1` which communicates with the user via an `I/O` object and an `Adapter` object. The type of the `Adapter` object must correspond to the type of the `I/O` object used (which depends on the terminal used). The initiation consists of the following steps:

1. An arbitrary object `CO2` of ***application XX*** invokes `CreateService(Out,`$User_a$`,ASI)` on `SF` (Service Factory).

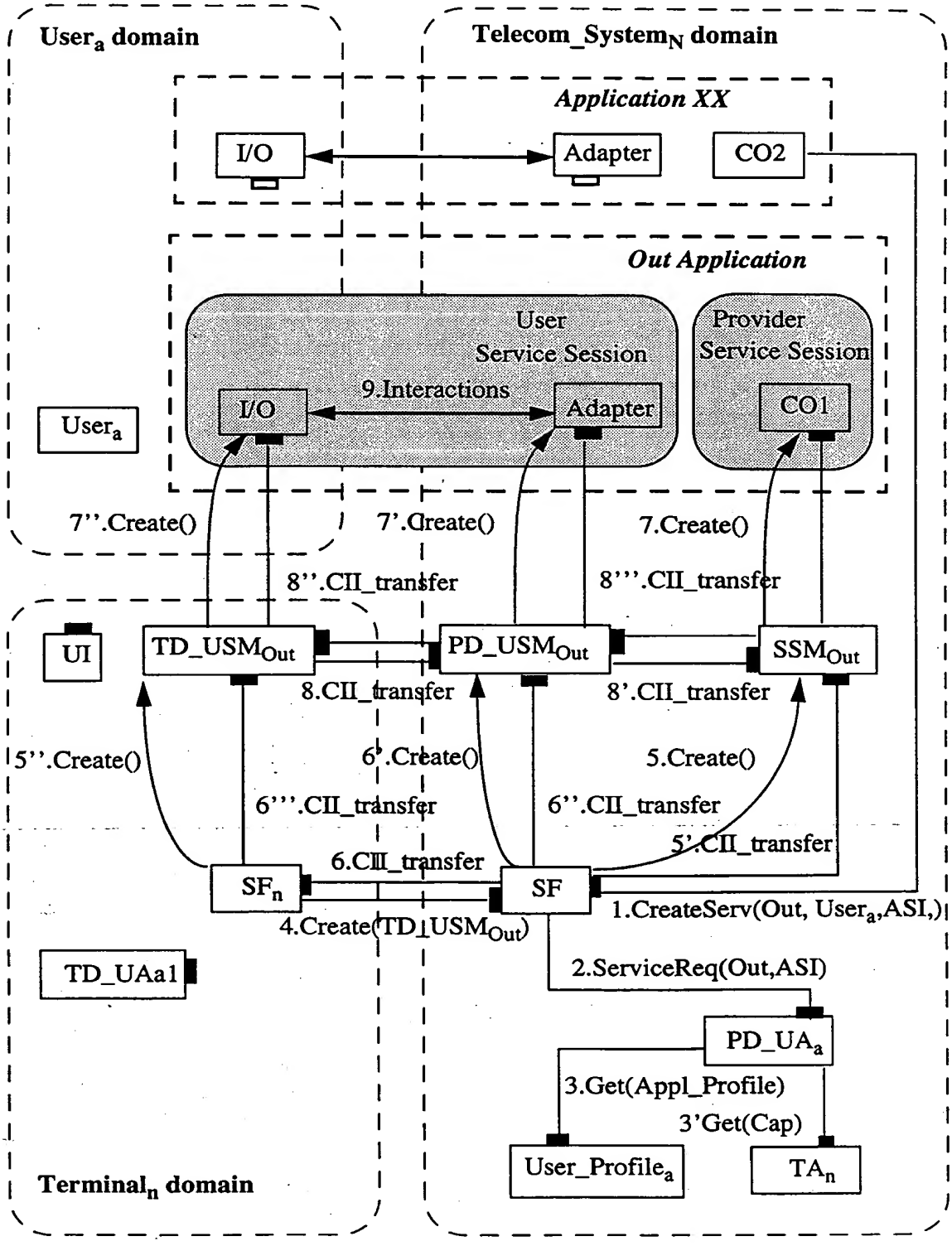


Figure 9.13 Initiation of a Generic Outgoing Application

2. Since the invocation is not requested by PD_UA_a, SF has to ask for permission and additional information form PD_UA_a. SF therefore invokes ServiceRequest(Out, ASI) on PD_UA_a.

Upon receipt of the invocation, PD_UA_a initiates the access control procedure of the ***out application*** in order to verify that $User_a$ has the right to use it (see Paragraph 7.4.3.3). If the access control procedure fails, PD_UA_a will return a negative acknowledgement and the initiation stops here. We suppose that the access control procedure is successful.

3. PD_UA_a invokes a Get(User_Appl_Profile,Out) to the $User\_Profile_a$ object in order to get the profile of the ***out application*** which contains $User_a$ defined data for the ***out application*** if such a profile is defined. Otherwise, the main User_Profile will be used to initiate the ***out application.***

3'. PD_UA_a invokes a Get(Capability) operation on TA_n to get the capability of the $Terminal_n$.

PD_UA_a returns the User_Appl_Profile and the Term_Capability in the response of the operation ServiceRequest to the SF.

4. SF deduces the CII of SF_n from the ASI (Access Session Identifier) and invokes $Create(TD\_USM_{Out})$.

5, 5', 5''. SF and SF_n create SSM_{Out}, PD_USM_{Out} and TD_USM_{Out}.

6, 6', 6'', 6'''. SF and SF_n transfer the CIIs of the created session manager objects to each other, e.g the CII of TD_USM_{Out} is transferred to PD_USM_{Out} and vice versa. The created session manager objects are now able to communicate with each other.

7, 7', 7''. The session manager objects create the objects belonging to their session. In our example, TD_USM_{Out} creates the I/O object, PD_USM_{Out} the Adapter and the SSM_{Out} the CO1 object. Recall that during the object creation, objects on the user domain will be registered to the PD_UA_a (see Chapter 9) and Proxy objects for objects having inter-domain communications will also be creates (see Chapter 11). The session manager objects are responsible for the initialization of all the application objects.

8, 8', 8'', 8'''. The session managers transfer the necessary CII of the created application objects to each other, e.g the CII of the I/O object is transferred to the Adapter object and vice versa.

8. From this step all the application objects are completely initialized and can operate properly. The initiation of a generic outgoing application is hence completed.

Note that for an outgoing application, mobility is transparent because the application does not need to interact directly with any object of the GMS. All operations be-

tween application objects residing in the user domain and the telecom system domain are converted and redirected by the proxy objects (not shown in the figure, see Paragraph 8.5) and conveyed by the GMS objects. The application designer therefore does not need to be aware of mobility and how the GMS is composed. The plant engineer, i.e the installer of the application in the telecom system domain, on the other hand, must, of course, have knowledge the GMS in order to implement the session manager objects (SSM_{Out}, PD_USM_{Out} and TD_USM_{Out}). Such a generic outgoing application can be classified as **Mobility-unaware applications**.

9.5.3 Incoming applications

9.5.3.1 Definition

An incoming application is intended to a user but is not initiated by him. It may be initiated by another user or by a service. An incoming application can be a telecommunication application such as telephony, data, facsimile, etc. or computing applications such as mail application, waking application, time manager applications which remind the user of different events, etc.

9.5.3.2 Information viewpoint

All the information necessary for the initiation of an incoming application is described by the information model shown in Figure 9.14. This information model is similar to the information model for service management of UPT presented in [Do96d] and the information model for an incall presented in [Aud96].

One **Incoming_Application** can be initiated at one or several **Current_Terminals**. Similarly, one or more **Incoming_Application** can be initiated at one **Current_Terminal**. The relation between **Current_Terminal** and **Incoming_Application** is called **Delivery_Restriction** and expresses the condition for delivery of the application at the current terminal. It is determined by the **Application_Restriction, Terminal_Capability** and **Usage_Restriction** objects. The **Usage_Restriction** information object is to allow the **Third_Party** to limit the use of his terminal and is used in the access control procedure in Chapter 7. If the **Current_Terminal** is determined, then the **Usage_Restriction** information object can also be found.

The crucial information for the initiation of an incoming application is the identity of the terminal to be used. The information about which **Current_Terminal** information object should used by the **Incoming_Application** is contained in one or two **Routing_Infos**. The first one and with the higher processing priority is the **Routing_Info** contained in the **User_Appl_Profile**. If such information is defined for the given application, this **Routing_Info** will be used to resolve the identity of the **Current_Terminal**. If it is not defined, then the **Routing_Info** contained in the main **User_Profile** is used for the resolution. This **Routing_Info** acts as a default **Routing_Info** for all the applications.

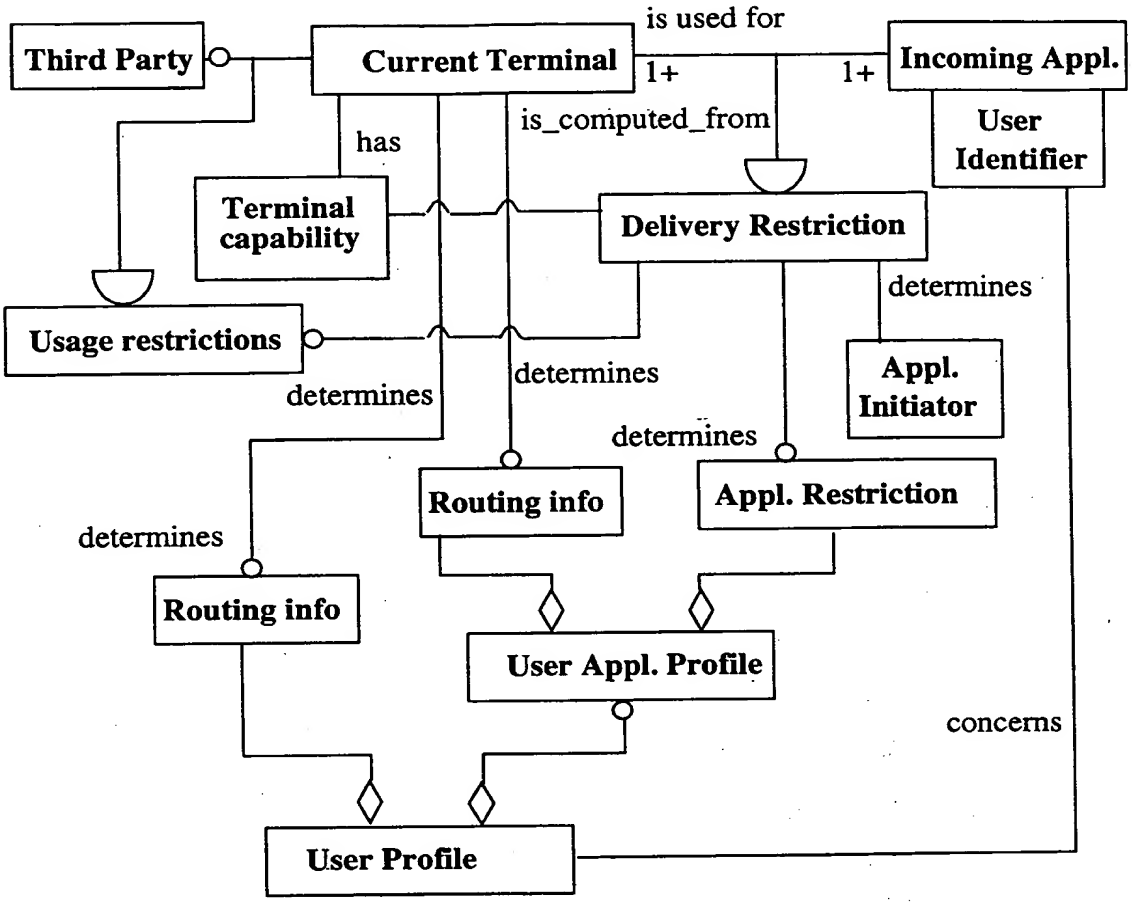


Figure 9.14 The information model for an Incoming application

Although the resolution of the **Current_Terminal** is based on multiple **Routing_Infos**, there is no guarantee that the **Current_Terminal** can be determined. Both **Routing_Infos** can be empty such as in the case where the user does not want to receive any incoming application or that a registration for a terminal does not exist at the time when the incoming application is activated. When the user wants to receive a specific Incoming application, he may have to do a registration for this incoming application. There may also be applications which do not require registration.

The registration of an incoming application is in fact an outgoing application since it is initiated by the user himself. It is therefore possible to conclude that every incoming application requires an outgoing application for its registration.

Before looking at a generic incoming application, let us first study briefly a generic registration application.

9.5.3.3 A generic registration application

There are multiple variants of registration. A registration can be done for all the subscribed incoming applications. The main **Routing_Info** of the **User_profile** is used in this case. It can be done for a specific incoming application. A **Routing_Info** is defined in the **User_Appl_Profile** of the application. It can also be done for a group of incoming applications while leaving out some specific ones. In this case, one **Routing_Info** is defined for each of the particular application. The registration of incoming applications can be linked to an access session, i.e every time the user logs in at a terminal the registration is automatically executed for the current terminal. In addition, as specified in Paragraph 9.3, the registration can also be done according to a time table, according to calling party identity, etc. How the registration is done is a matter of policy and implementation and is not relevant for our discussion. Our goal is to define a generic registration in such a way that all the registration variants are subtypes of the generic one.

Although a registration application is an outgoing application, it has a major difference compared with the outgoing application we described in Paragraph 9.5.2. As noted there the generic outgoing application is mobility-unaware. A registration application is dealing with the updating of the location of the user. A registration application can therefore not be mobility-unaware. Indeed, it has to interact with the GMS and more specifically the `PD_UA` object in order to accomplish its mission.

Since a registration application is an outgoing application, its initiation is therefore identical to the initiation of a generic outgoing application presented in Paragraph 9.5.2 and will not be considered again. We shall only look at the interface between a registration application and the GMS, i.e what additional operations are required with the GMS in order to support registration.

Let us consider a generic registration application for an incoming application XX consisting of an object `CO1`, an `Adapter` object and an `I/O` object as shown in Figure 9.15. Suppose that the registration application is successfully initialized and running properly. `CO1` is the object responsible for the registration. The following steps will be executed:

1. `CO1` invokes the operation `Get(Routing_Info, XXid)` on the $\texttt{PD\_UA}_a$ object `XXid` is the identifier of the application XX.

1'. $\texttt{PD\_UA}_a$ invokes the operation `Get(Routing_Info, XXid)` on the $\texttt{User\_Profile}_a$ object. If there is a specific `Routing_Info` defined for application XX, it will be returned to $\texttt{PD\_UA}_a$. Otherwise, the default `Routing_Info` contained in the `User_Profile` object will be returned to $\texttt{PD\_UA}_a$. $\texttt{PD\_UA}_a$ transfers the `Routing_Info` back to `CO1`.

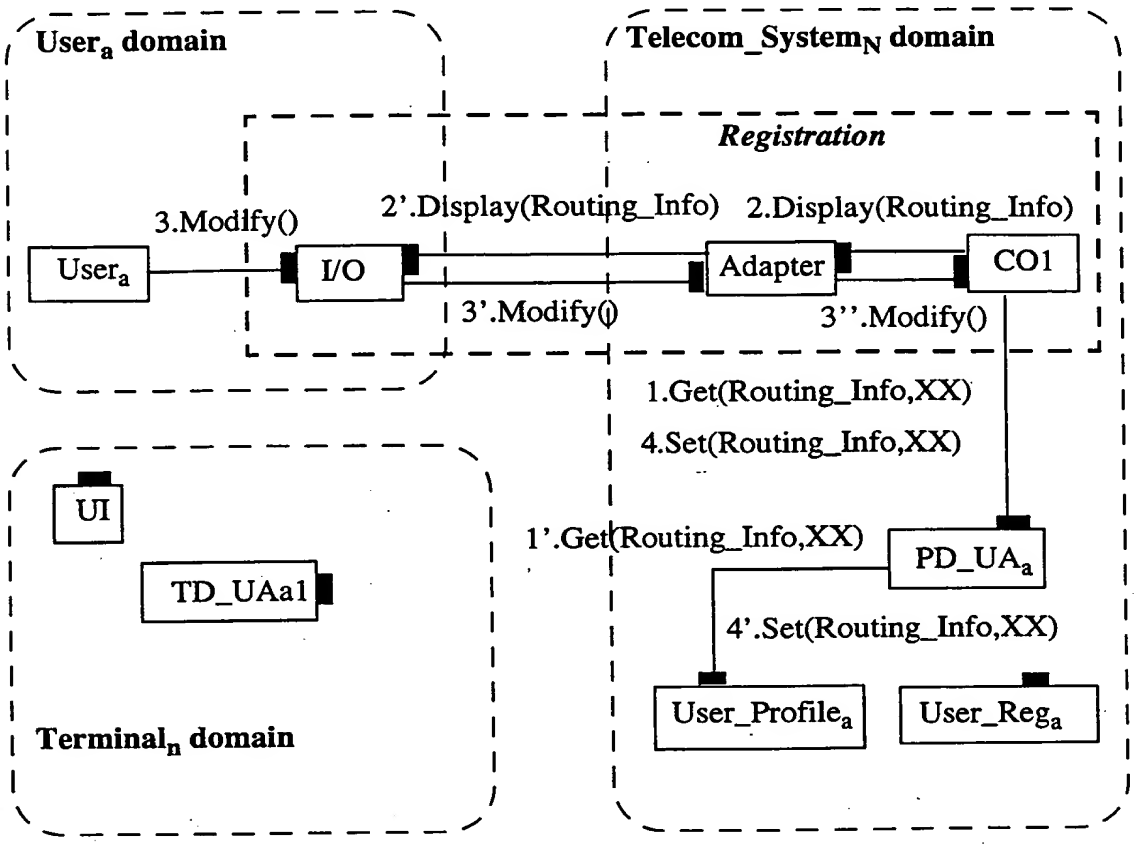


Figure 9.15 Interface between Registration application and the GMS

2, 2'. `CO1` invokes the operation `Display(Routing_Info)` on the `Adapter` object which invokes the same operation on the `I/O` object. The `Routing_Info` is then displayed to **User$_a$**. The way in which it is displayed is a matter of implementation.

3, 3', 3''. The `Routing_Info`, as modified by **User$_a$**, is conveyed back to `CO1`.

4, 4' `CO1` invokes `Set(Routing_Info, XXid)` on **PD_UA$_a$** in order to save the modified `Routing_Info`. **PD_UA$_a$** invokes the same operation on **User_Profile$_a$**. The `Routing_Info` is now saved and ready for use. This concludes the registration of an application.

9.5.3.4 A generic incoming application

To support the creation and delivery of an incoming application, an additional computational object called `Locator` is added in the GMS. The `Locator` object will assists the **PD_UA$_a$** object in locating User$_a$. The `Locator` will process the data con-

tained in the Routing_Info contained in the User_Appl_Profile or in the main Routing_Info and find the CII (Computational Interface Identifier) of the TA object representing the terminal at which the incoming application should be delivered.

There are several ways of computing the delivery address and they depend on the preference of the user. To clarify this, recall that a Routing_Info may have the following attributes:

- Forwarding activation status
- Forwarding terminal address
- Registered terminal address for incoming applications
- Linked-registered terminal address
- Default terminal address for incoming applications
- Routing by applications originating area
- Routing by calling party identity
- Time-dependent routing
- Routing on "busy" condition
- Routing on "no answer"

It is evident that the forwarding address, when defined, should have the highest priority and then comes the registered address with second highest priority. However, it is not evident how the routing by applications originating area, routing by calling party identity, time-dependent routing should be ranked. A user may give priority to the routing by calling party identity while another one prefers the time-dependent routing.

The GMS must ensure flexibility and allow all alternatives. A **Locator** implementing all the different location resolution algorithms is therefore proposed and, depending on the preference of the user, one specific algorithm will be chosen and used. There may be several instances of the **Locator** in the telecom system domain. The detailed implementation of such a **Locator** falls beyond the scope of this thesis and will not be considered further.

The **Locator** will be equipped at least with the following operation specified as follows using IDL:

```
Resolve(in Routing_Info,in Status,in Preference,out TAI)
```

Routing_Info contains routing data and is either a component of the User_Appl_Profile or the User_Profile. Status is used to indicate whether the resolution request is issued for the first time or it is re-issued because the user is busy or does not answer. Different processing will be carried out depending on the value of Status which can be First, Busy or NoAnswer. Preference contains information on the priority ranking of the different routing alternatives. TAI is the CII of the current TA and will be returned as result to the invocation.

Let us now study the initiation of a generic incoming application called ***In application*** intended to a $User_a$ who has previously registered himself for the ***In application*** at a $terminal_n$. ***In application*** is initiated by an arbitrary outgoing application, say ***XX*** which itself is not initiated by $User_a$ (Otherwise the ***In application*** will be not be an incoming application but an outgoing one). Figure 9.16 shows two ways to initiate an incoming application. In case a) the incoming application is started by an outgoing application which is initiated by another user, $User_b$. In case b) the incoming application is started by an application which is initiated by the telecom system domain, e.g waking application.

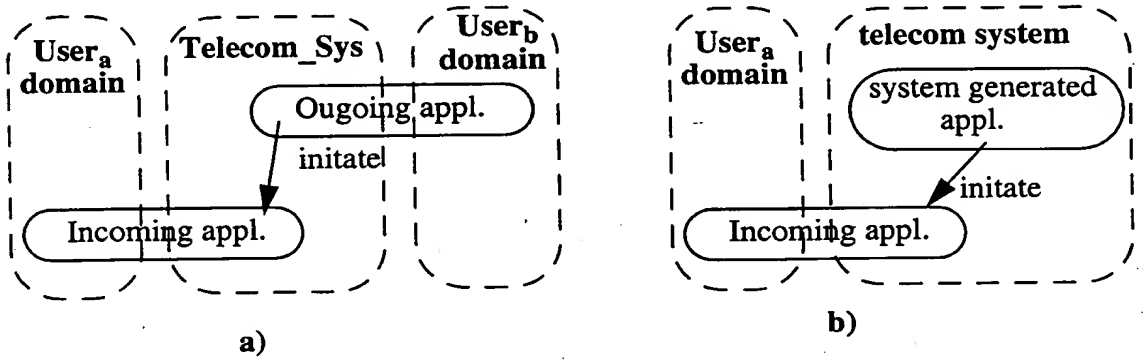


Figure 9.16 Two ways to initiate an incoming application

For the sake of simplicity and without losing generality, let suppose that ***In application*** consists of one object `CO1` and communicates with the user via an `I/O` object and an `Adapter` object. The type of the `Adapter` must correspond to the type of `I/O` object which again may depend on the terminal used.

Steps 1 to 5 of the initiation are depicted in Figure 9.17.

1. `CO2` of ***application XX*** invokes `CreateService(In,`$User_a$`)` on `SF` (Service Factory).

2. Since the invocation is not requested by PD_UA_a, `SF` has to ask for permission and additional information from PD_UA_a. `SF` invokes `ServiceRequest(In)` on PD_UA_a.

Upon receipt of the invocation, PD_UA_a discovers that the requested service session is no yet assigned to an access session since no ASI (Access Session Identity) is given in the call. It has to find the identity of the terminal at which the service session should be delivered.

3. PD_UA_a invokes the operation `Get(Routing_Info,In)` on the $User\_Profile_a$. If there is a `Routing_Info` defined specifically for the given

application, it will be returned to PD_UA_a. Otherwise, the default `Routing_Info` will be returned to PD_UA_a.

4. PD_UA_a invokes the operation `Resolve(Routing_Info, First, Preferences)` on the `Locator` object.

If $User_a$ has no registration for ***In application*** (he may still be allowed to use outgoing applications), a Nil is returned in TAI to PD_UA_a. PD_UA_a returns an unsuccessful status to `SF` which in turn informs `CO2`, indicatinf that it is not possible to deliver the ***In application*** for $User_a$ for the time being.

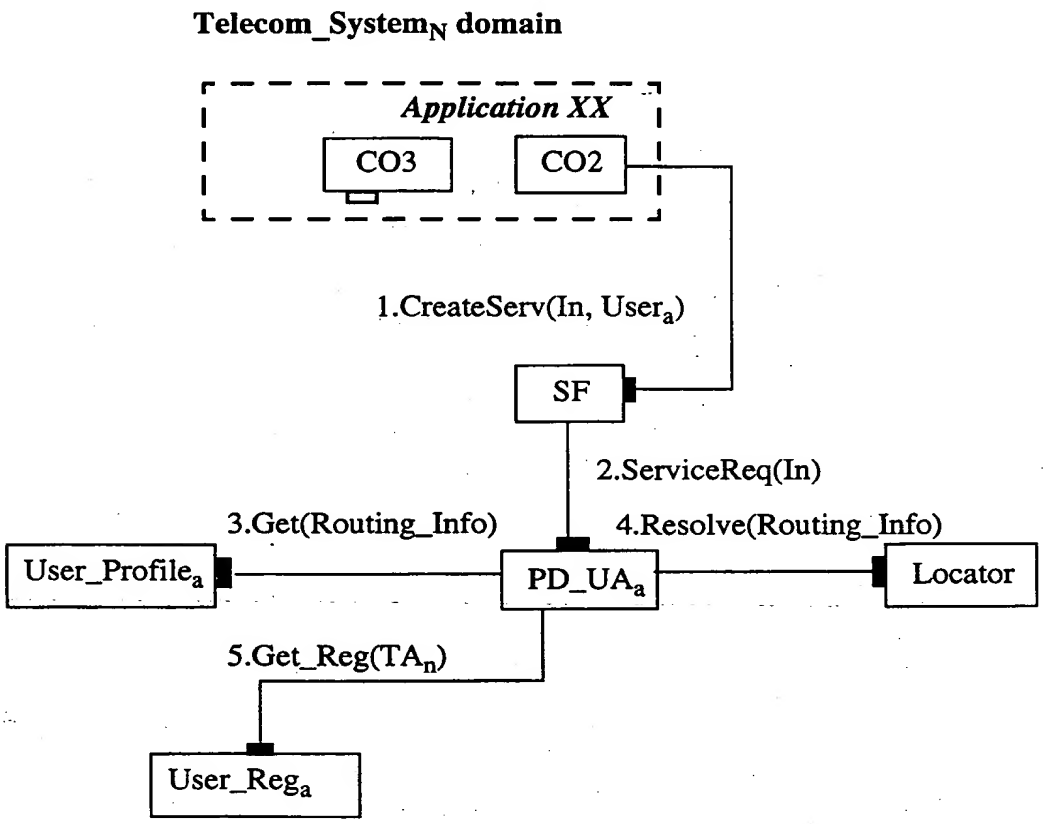


Figure 9.17 Steps 1 to 5 of the initiation of an Incoming application

If $User_a$ has registered for ***In application***, the `Locator` will return to PD_UA_a a `TAI` with value identifying TA_n object.

5. PD_UA_a invokes the operation `Get_Registration(`TA_n`)` on the $User\_Registration_a$ object to get all the data related to the usage at $Terminal_n$, i.e a row containing `TAI, TD_UAid` in the table of the $User\_Registration_a$ object (See Figure 9.5).

From this step there are two cases that should be considered separably:

a. $User_a$ has already logged in at $Terminal_n$. A TD_UA_{ax} object and an access session exists already on $Terminal_n$.

b. $User_a$ has not logged in at $Terminal_n$. Neither a TD_UA_{ax} nor an access session exists for the user at $Terminal_n$.

Case a - $User_a$ has logged in:

6. PD_UA_a performs access control on the requested service carrying the ***In Application*** (See Paragraph 10.4.3.3). If this access control procedure fails, PD_UA_a will return a `NotAllowed` status to `SF` which will inform ***application XX***. This step is shown as an encircled 6 in Figure 9.18

7. If the access control procedure succeeds, PD_UA_a invokes a `Get(User_Appl_Profile, In)` on the $User\_Profile_a$ object to get the profile of the ***In application*** if such a profile is defined the application. Otherwise, the default profile will be used to set up the ***In application***.

8. PD_UA_a invokes a `Get(Capability)` operation on TA_n to get the capabilities of $Terminal_n$.

9. PD_UA_a assigns the requested Service session to the current access session and returns to `SF` in the response of the operation `ServiceRequest` an `Allowed` status, the ASI (Access Session Identifier) of the active access session, the `User_Appl_Profile`, and the `Term_Capability` . The response is denoted as `RServiceRequest` in Figure 9.18.

10. `SF` deduces the CII of SF_n from the ASI (Access Session Identifier) and invokes `Create(`TD_USM_{In}`)`.

11, 11', 11''. `SF` and SF_n create the session manager objects SSM_{In}, PD_USM_{In} and TD_USM_{In}.

12, 12', 12'', 12'''. `SF` and SF_n exchange and transfer the CIIs of the created session manager objects to each other, e.g the CII of TD_USM_{In} is transferred to PD_USM_{In} and vice versa. The created session manager objects are now able to communicate with each other.

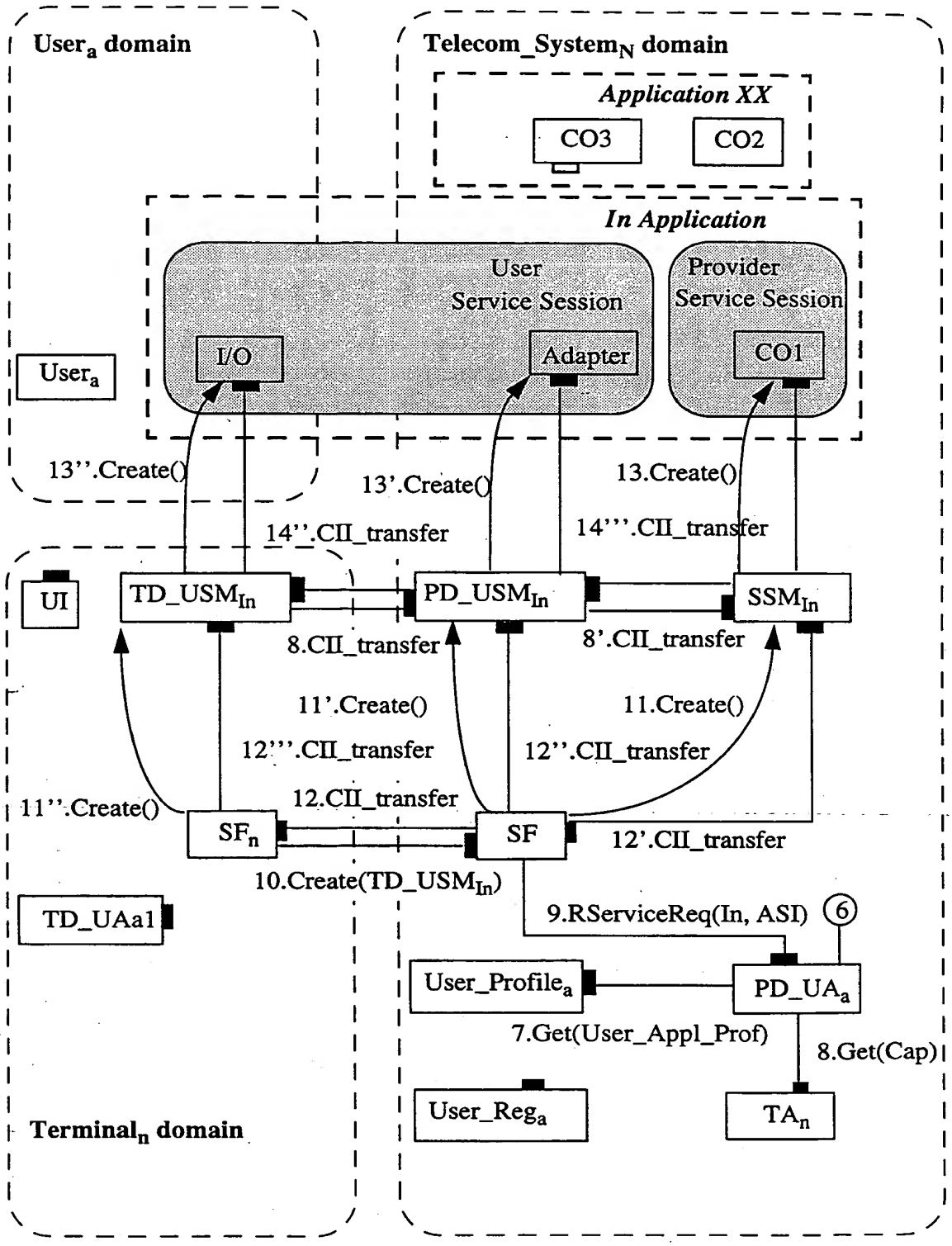


Figure 9.18 Step 6 to 14 of the initiation in case a

13, 13', 13''. The session manager objects create the objects belonging to their session. In our example TD_USM_{In} creates the I/O object, PD_USM_{In} creates the Adapter and the SSM_{In} creates the CO1 object. Recall that during the object creation, objects on the user domain will be registered with the PD_UA_a (see Chapter 6) and rroxy objects for objects having inter-domain communications will also be creates (see Chapter 8). The session manager objects are responsible for the initialization of all the application objects.

14, 14', 14'', 14'''. The session manager objects exchange and transfer the necessary CII of the created application objects to each other, e.g the CII of the I/O object is transferred to the Adapter object and vice versa.

15. The SSM_{In} object invokes the operation Output(Notification) on the PD_USM_{In} object to start the notification towards $User_a$. This step and the following ones are shown in Figure 9.19.

15'. The SSM_{In} sets a timer.

16. There are many ways in which to notify or alert the user concerning the arrival of an incoming application. The method will depend on the capability of the terminal, e.g ringing, icon blinking, alert windows, etc. The PD_USM_{In} object will choose the appropriate notification method and invokes Output(Notification) on the TD_USM_{In} object.

17. TD_USM_{In} invokes in its turn Output(Notification) on the I/O object.

18. The I/O object notifies $User_a$.

19. If $User_a$ is present and responds to the alert provided by the I/O object, the I/O object will proceed to deliver its service to $User_a$. The initiation of ***In application*** is then successfully accomplished.

On the other hand, if $User_a$ does not answer within a certain time period, the timer will inform the SSM_{In} object.

20. SSM_{In} invokes again the operation ServiceRequest(In) on PD_UA_a.

21. Since the ServiceRequest(In)originates form the SSM_{In} and not from the SF, that PD_UA_a knows that the ***In application*** has been invoked before. It invokes therefore invokes the operation Get(Routing_Info,In) on the $User\_Profile_a$. If there is a Routing_Info defined for the incoming application, this will be returned to PD_UA_a. Otherwise, the default Routing_Info will be returned to PD_UA_a.

22. PD_UA_a invokes the operation Resolve(Routing_Info,NoAnswer, Preferences) on the Locator, to get a new alternative delivery address, i.e new TAI (Terminal Agent Identifier).

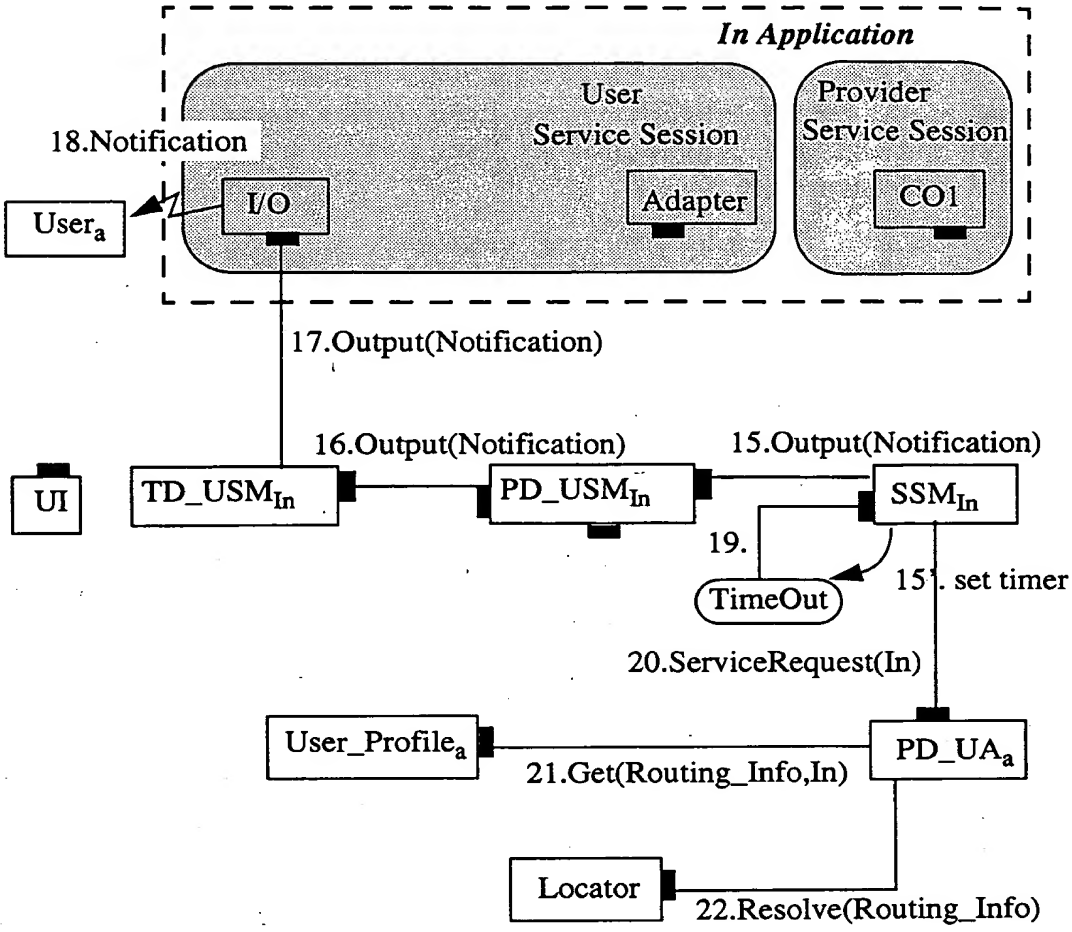


Figure 9.19 Step 15 to 22 of the initiation in case a

From this step onwards, all the procedures i.e access control on the requested service, access control for access to the telecom system domain, usser registration and access control for use of the current terminal will be executed again. Depending on the result of each procedures, the initiation will proceed with case a or case b. Since the operation sequence is quite similar to the one described before, it will not be repeated here. The only point worth to be mentioned is that only the $\texttt{TD\_USM}_{\texttt{In}}$ and $\texttt{PD\_USM}_{\texttt{In}}$ will be replaced by the new instances created for the new terminal while the $\texttt{SSM}_{\texttt{In}}$ remains unchanged.

The initiation will be repeated either until User$_a$ is found or until the location process fails. The failure criterion may be that all routing possibilities have been exhausted. The description of the initiation of case a of the generic incoming application is hence completed.

Case b - $User_a$ not logged in:

6. In this case, it is not known whether or not $User_a$ has access rights to the telecom system domain. PD_UA_a will perform the access control for access to the telecom system domain (See Paragraph 7.4.3.2). This step is shown as an encircled 6 in Figure 9.20. If this access control procedure fails, PD_UA_a will return a `NotAllowed` status to **SF** which will inform ***application XX***. The initiation of ***In Application*** terminates here. Otherwise, it will continue with the next step.

7. PD_UA_a performs the access control on the requested service carrying the ***In Application*** (See Paragraph 7.4.3.2). This step is shown as an encircled 7 in Figure 9.20. If this access control procedure fails, PD_UA_a will return a `NotAllowed` status to the **SF** which will inform ***application XX***.

8. If the access control succeeds, PD_UA_a performs the remote user registration procedure by invoking `Register(`$User_a$`)` on TA_n as described in Paragraph 6.4.3.5. The access control for the use of $Terminal_n$ is thereafter executed.

If this access control procedure fails, PD_UA_a will return a `NotAllowed` status to **SF** which will inform ***application XX***. The initiation of ***In Application*** terminates here.

If the access control succeeds, a TD_UA_{ax} is created in $Terminal_n$. The $User\_Registration_a$ object has also created a new row containing the identifiers of TA_n and TD_UA_{ax}.

9. PD_UA_a invokes `AccessReg(ASI,Suspended,TAn)` on the $User\_Registration_a$ object to create and register an access session at $Terminal_n$. The state of this access session is `Suspended` in this case because the user is not yet authenticated.

10. PD_UA_a invokes a `Get(User_Appl_Profile,In)` to the $User\_Profile_a$ to get the profile of the ***In application*** if such a profile is defined. Otherwise, the default User_profile will be used to set up the ***In application***.

11. PD_UA_a invokes a `Get(Capability)` operation on TA_n to get the capabilities of $Terminal_n$.

12. PD_UA_a returns an `Allowed` status, the `ASI` (Access Session Identifier) of the access session plus the state value `Limited`, the `User_Appl_Profile`, and the `Term_Capability` in the response of the operation `ServiceRequest` to the **SF**. The state value `Limited` means that the access session is allowed only to display information to the user but not to read input from the user because the user has not yet been authenticated.

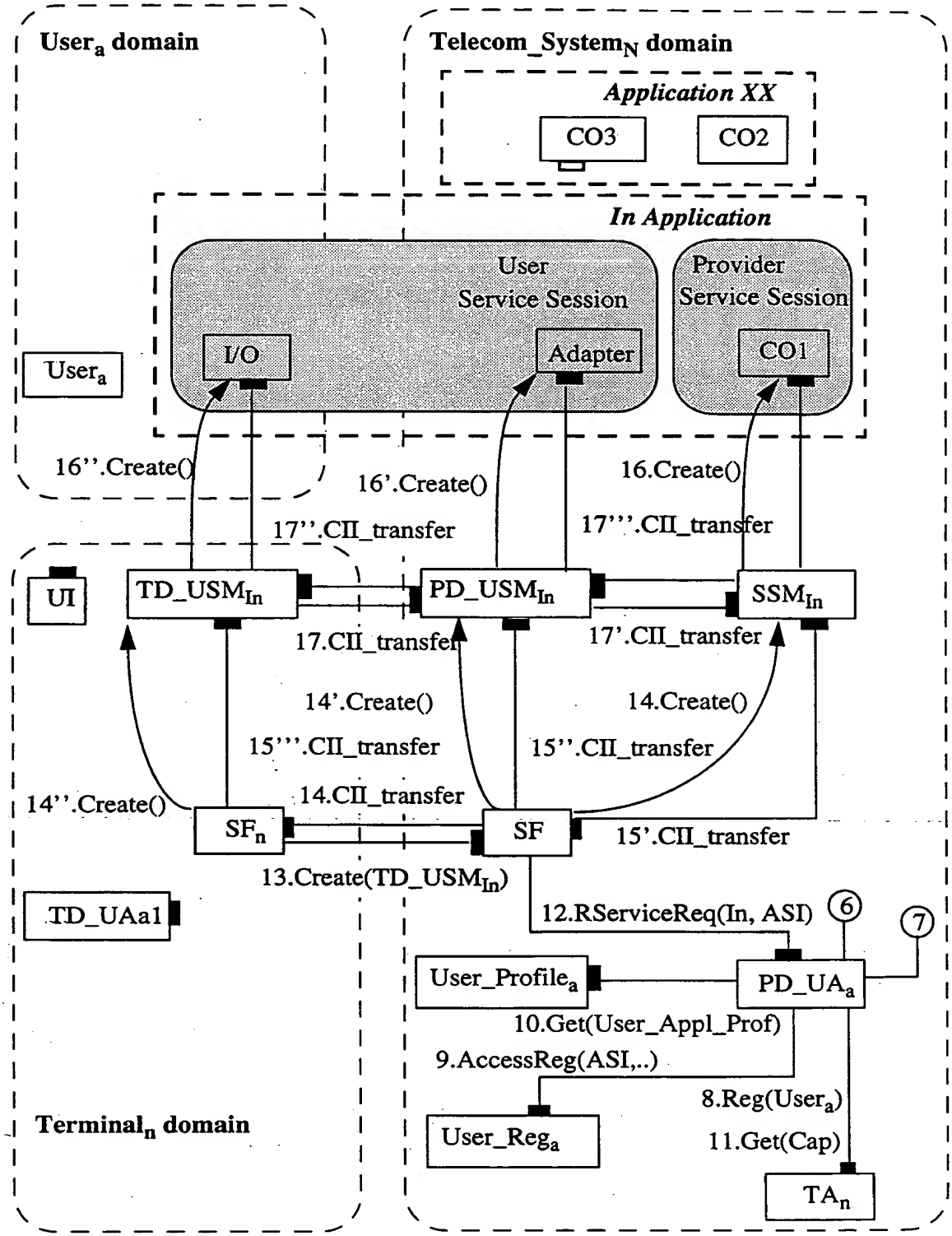


Figure 9.20 Step 6 to 17 of the initiation in case b

13. SF deduces the CII of SF_n from the ASI (Access Session Identifier) and invokes Create(TD_USM_{In}).

14, 14', 14''. SF and SF_n create the session manager objects SSM_{In}, PD_USM_{In} and TD_USM_{In}.

15, 15', 15'', 15'''. SF and SF_n exchange and transfer the CIIs of the created session manager objects to each other. The created session manager objects are now able to communicate with each other.

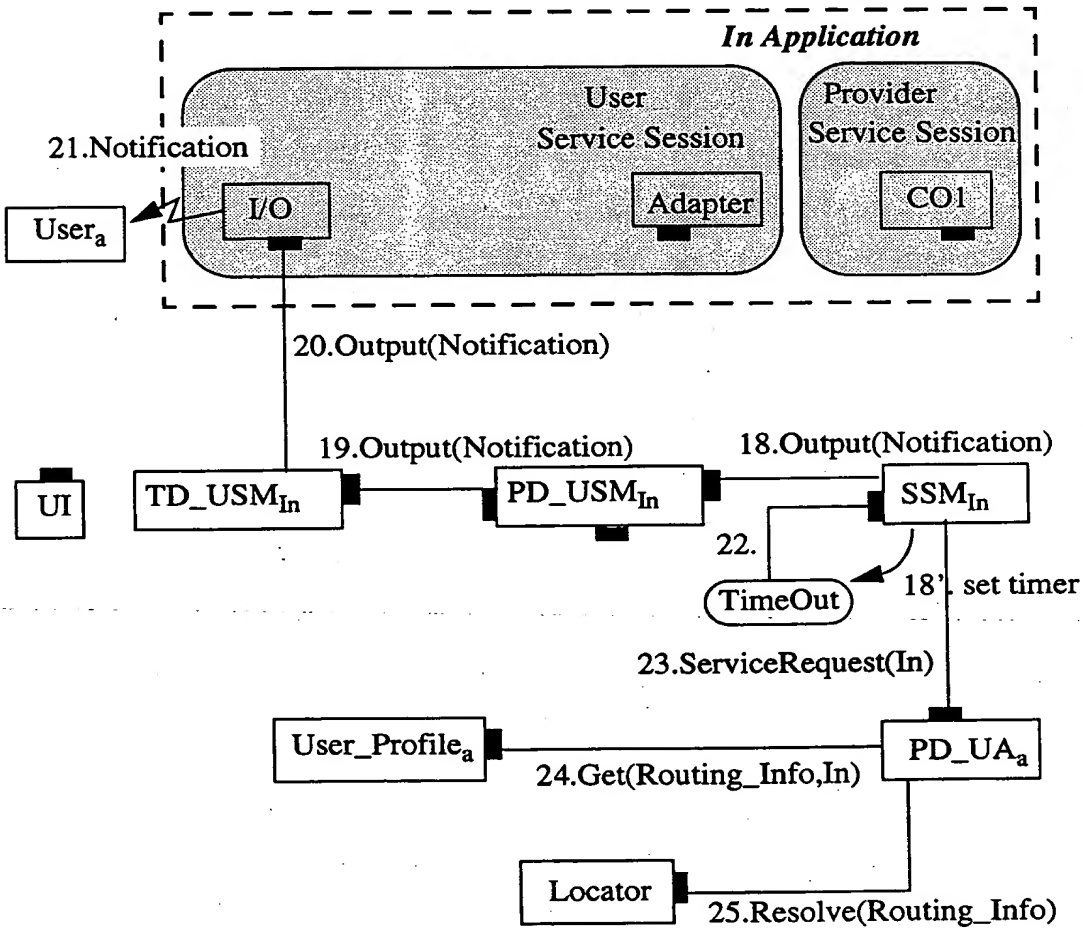


Figure 9.21 Step 18 to of the initiation of case b

16, 16', 16''. The session manager objects create the objects belonging to their session. In our example, TD_USM_{In} creates the I/O object, PD_USM_{In} the Adapter

and the SSM_{In} the CO1 object. It is worth to recall that during the object creation, objects on the user domain will be registered to the PD_UA_a (see Chapter 9) and Proxy objects for objects having inter-domain communications will also be creates (see Chapter 11). The session manager objects are responsible for the initialization of all the application objects. Since the state of the Service session is Limited, the I/O object is set to the OutputOnly mode, i.e it can display information to the user but can not receive input from the user.

17, 17', 17'', 17'''. The session manager objects exchange and transfer the necessary CII of the created application objects to each other, e.g the CII of the I/O object is transferred to the Adapter object and vice versa.

18. The SSM_{In} invokes the operation Output(Notification) on the PD_USM_{In} to start alerting to $User_a$. This step and the following ones are shown in Figure 9.21.

18'. The SSM_{In} sets a timer.

19. The PD_USM_{In} will choose the appropriate notification method and invokes Output(Notification) on the TD_USM_{In}.

20. TD_USM_{In} invokes in its turn Output(Notification) on the I/O object.

21. The I/O object notifies $User_a$.

22. If $User_a$ is present and wants to receive the incoming application, he has to log himself in at the $Terminal_n$ and initiate an access session. He can not respond directly to the I/O object because it is in OutputOnly mode. After an access session has successfully been created, $User_a$ can interact with the I/O object to receive the ***In application***.The initiation of ***In application*** is hence successfully accomplished.

On the other hand, if $User_a$ does not answer within a certain time period, a timer will inform the SSM_{In}.

23. SSM_{In} invokes again the operation ServiceRequest(In) on PD_UA_a.

24. Upon receipt of the invocation, PD_UA_a discovers that the ServiceRequest(In) has been invoked before. It therefore invokes the operation Get(Routing_Info,In) on the $User\_Profile_a$. If there is a Routing_Info defined for the incoming application, this will be returned to PD_UA_a. Otherwise, the default Routing_Info contained in the user profile information object will be returned to PD_UA_a.

25. PD_UA_a invokes the operation Resolve(Routing_Info,NoAnswer, Preferences) on the Locator object in order to get a new alternative delivery address, i.e new TAI (Terminal Agent Identifier).

From this step onwards, all the procedures i.e access control on the requested service, access control for access to the telecom system domain, user registration and access control for the use of current terminal will be executed again. Depending on the result of each procedures, the initiation will proceed with case a or case b. Since the operation sequence is quite similar to the one described before, it will not be repeated here. Note, however, that the TD_USM_{In} and PD_USM_{In} will be replaced by the new instances created for the new terminal while the SSM_{In} remains unchanged.

The initiation will be repeated until $User_a$ is found or until the location process fails. The description of the initiation of the case b of a generic incoming application is hence completed.

Note that in our description of the procedure we have ssaumed that all the steps have to be performed in series. However, several of them (e.g those associated with access control) may be performed in parallel as linked transactions using normal commit and recovery procedures. Use of linked transactions may speed up the overall procedure. The mechanisms are, however, ratheer complex and may require additional transaction control objects. We will not consider this issue further in this thesis.

9.6 SUPPORTING SESSION MOBILITY

9.6.1 Definition

The same definition of session mobility as TINA-C [TIN95j] is used in this thesis:

"Session mobility facilitates a service session, being used by a user, to "follow" that user, independently of the location of the user, or of the terminal, or of access point to the network."

9.6.2 Session mobility support

Although the definition of session mobility used in this thesis is identical with the one of TINA-C, the proposed solution to support session mobility is fundamentally different to the one of TINA-C presented in [TIN95j].

Let us suppose that a $User_a$ is using an application A which can be either outgoing or incoming at $Terminal_n$. $User_a$ wants now to move to another terminal, say $Terminal_m$ without disrupting application A. He must then suspend the service carrying application A, log in or start a new session at the new terminal, and resume application A at the new terminal. Note that $User_a$ may or may not log out of $Terminal_n$. He may have other applications running at $Terminal_n$which he wants to maintain. Note that there is always a Main application running after a successful access (see Paragraph 9.5.2.3).

As shown in Figure 9.22, the sequence of operations related to the suspension of a service session is as follows:

1. $User_a$ issues a Service Session Suspension request through the ***Main Application***. An object `CO2` of the ***Main application*** invokes the operation `SuspendService(A, ASI)` on PD_UA_a.

As specified in Paragraph 9.5.2.3, the ***Main application*** can be built with functionality to suspend and resume service sessions. How the user actually enters his request is a matter of implementation which is not relevant to our context. what we need is to find all the functions that the GMS and its objects must have in order to support session mobility.

2. PD_UA_a invokes the operation `SSModify(A,ASI,Suspended)` on the $User\_registration_a$ object to change the state of the service session carrying application A from `Active` to `Suspended`.

3. PD_UA_a invokes the operation `Suspend()` on the SSM_A object asking it to change from `Active` state to `Suspended` state.

4. SSM_A suspends all the application objects belonging to the provider service session by invoking `Suspend()`.

5, 5'. SSM_A invokes `Terminate()` on the PD_USM_A and the TD_USM_A objects.

6, 6'. The PD_USM_A and the TD_USM_A terminate all the application objects belonging to the terminal domain and Provider Domain User Service Session.

The suspension of the Service session carrying application A is now accomplished. $User_a$ can but does not have to, log out from $Terminal_n$. In the case that he logs out, the access session assigned to $Terminal_n$ will be terminated and removed from the table of the $User\_Registration_a$. The Service session of A which initially belongs to the access session, must however be saved in the `Suspended` state and with no access session supporting it. This is the special case where a Service Session persists when the owning access session terminates.

$User_a$ can now move to $Terminal_m$ and proceed with the activities to establish an access session (See Paragraph 9.4) or start with step 1. below if he is already running an access session on that terminal. We suppose that all the procedures are successful and a Main application is running on $Terminal_m$. $User_a$ can now request the resumption of the Service session carrying application A. The sequence of operations is shown in Figure 9.23.

1. $User_a$ issues a service session resumption request through the ***Main Application***. An arbitrary object `CO2` of the ***Main application*** invokes the operation `ResumeService(A, ASI)` on PD_UA_a.

2. PD_UA_a invokes `SSGet(A)` on the $User\_Registration_a$ object to get information concerning service session A (e.g CII of the SSM_A).

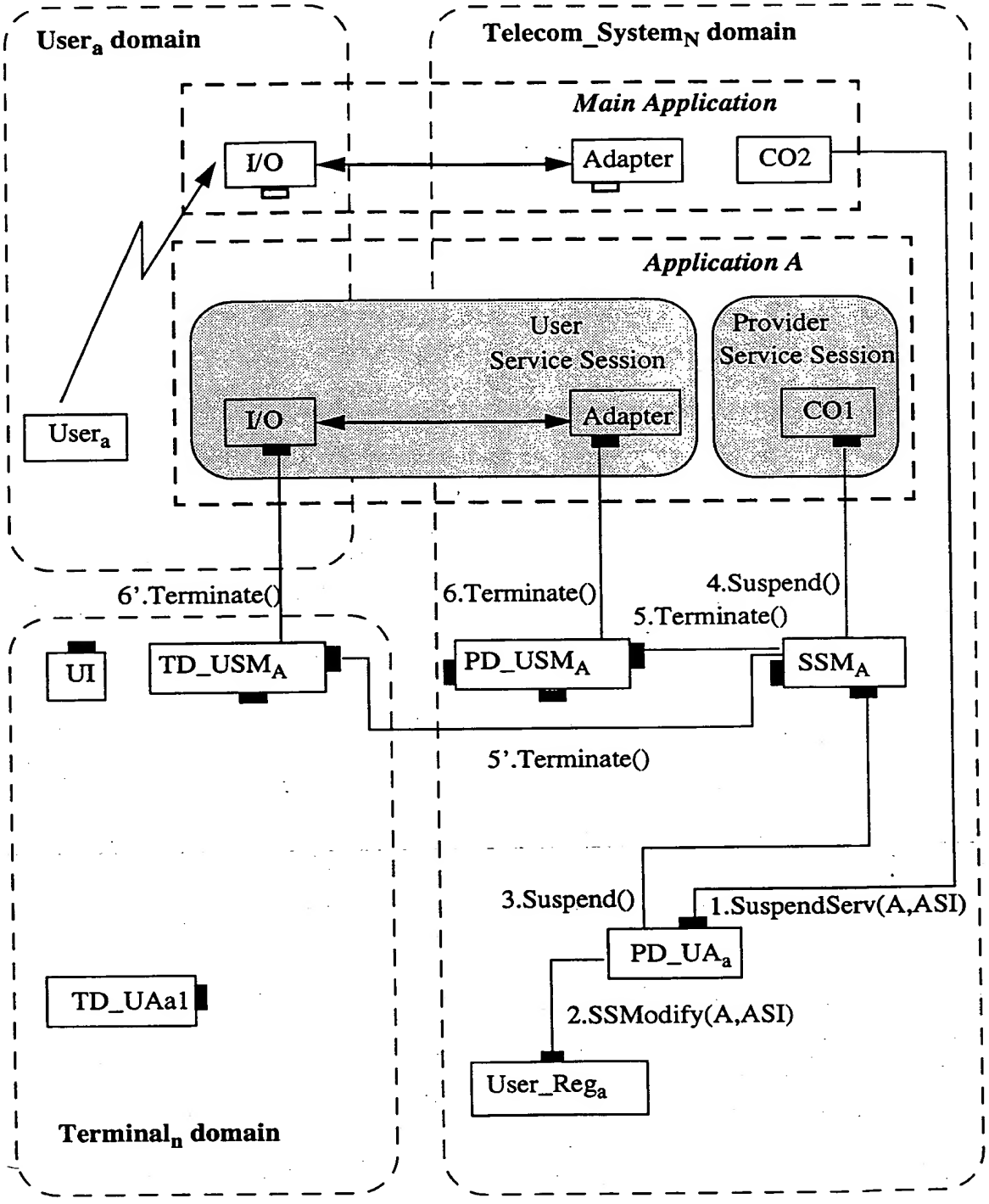


Figure 9.22 The suspension of a service session carrying an application A

3. PD_UA_a initiates the access control on the ***application A*** (for more details, see Paragraph 7.4.3.3). If the access control procedure fails, PD_UA_a will return a negative acknowledgement to CO2 and the resumption stops here. We suppose that the access control procedure is successful.

4. PD_UA_a invokes a `Get(Capability)` operation on TA_m to get the capability of the $Terminal_m$.

5. PD_UA_a invokes `Resume(ASI, Term_Capability)` on the SSM_A

6. SSM_A invokes `CreateUSM(A, User`$_a$`, ASI)` on the SF.

7. SF deduces the CII of SF_n from the ASI (Access Session Identifier) and invokes `Create(TD_USM`$_A$`)` on SF_n.

8, 8'. SF and SF_n create the session manager objects PD_USM_A and TD_USM_A.

9, 9', 9'', 9'''. SF and SF_n exchange and transfer the CIIs of the created session manager objects to each other, e.g the CII of TD_USM_A is transferred to PD_USM_A and vice versa. The created session manager objects are now able to communicate with each other.

10, 10'. TD_USM_A creates the I/O object and PD_USM_A creates the Adapter object. Recall that during the object creation, objects in the user domain will be registered with the PD_UA_a (see Chapter 6) and proxy objects for objects having inter-domain communications will also be creates (see Chapter 8). The session manager objects are responsible for the initialization of all the application objects.

11, 11', 11'', 11'''. The session manager objects transfer the necessary CIIs of the created application objects to each other, e.g the CII of the I/O object is transferred to the Adapter object and vice versa. All objects will then be set to `Active` state.

From this step onwards, all the application objects are completely initialized and can operate properly. The ***application A*** is now available to $User_a$ at the new terminal in the same state as before suspension. The description of the resumption of a service session carrying ***application A*** is hence completed.

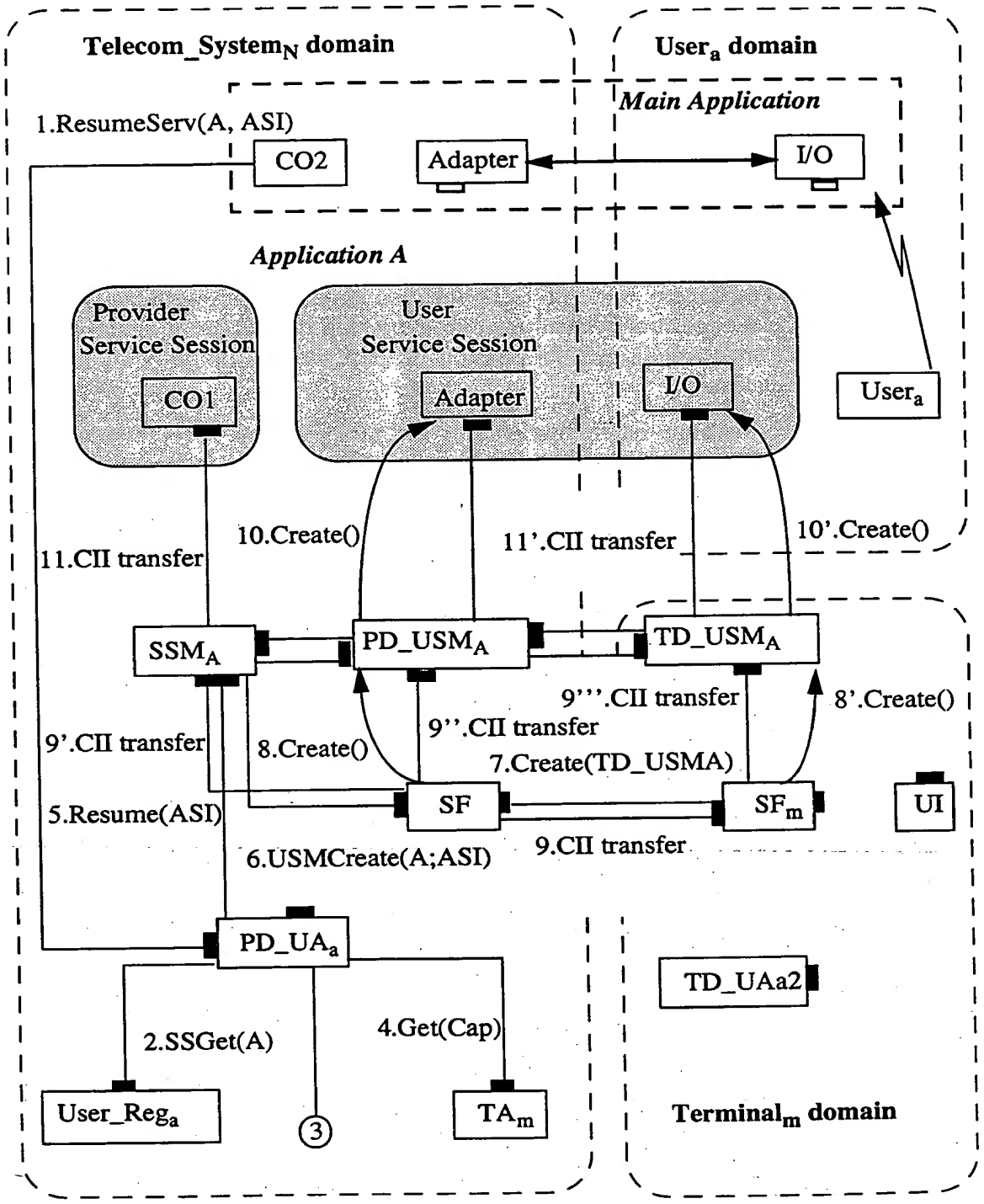


Figure 9.23 The resumption of a Service session carrying an application A

9.7 OFFERING SERVICES TO MOBILITY-BASED APPLICATIONS

Before leaving the topic of application management, we shall consider a new type of applications that we call mobility-based applications. We shall study what services they need and how the GMS may provides them.

9.7.1 Definition

A mobility-based application is a type of application which is born with the introduction of mobility in the telecom system domain and which uses actively mobility related information in the realization of its mission. In fact, without mobility such an application is meaningless.

Examples of such mobility-based application are taxi dispatch, fleet management, public safety, trucking, etc. Other mobility-based applications are information services to mobile users. For instance, traffic information or weather reports will be filtered based upon the current position of the user, while stock information will be filtered using the user profile. Another example of information services is a local Yellow-pages service extended with on-line information such as movies currently playing at local theatres or merchandise on sale at the local supermarket [IB94].

9.7.2 What services do they need?

A common requirement for all the mobility-based application is to obtain the location information of a user or a group of users. This information is used in further processing and decision processes which are application specific.

To clarify this, let us take the example of taxi dispatch. When the "real-time" location of all the taxis are known, it is possible to identify the available taxi closest to the customer.

9.7.3 Services to mobility-based application

The GMS, in order to support terminal and user mobility, does hold some form of location information. Although the GMS's location information may not correspond totally to the location information needed by the mobility-based application, they may be transformed and become useful.

The GMS has the following capabilities:

1. At any time and for any user the GMS can find the terminals the user is currently using and the terminals at which he wants to receive incoming applications if there is any.

2. At any time and for any terminal, the GMS can find the NAP coverage area where the terminal is currently located. (The NAP coverage area is discussed in Chapter 5)

3. From 1 and 2 above it is possible to deduce: at any time and for any user the GMS can find the NAP coverage area in which the user is currently located (if there is any).

If the NAP coverage area is very large (e.g city size) then the information about which NAP the user can currently be reached, may not be relevant for some applications, e.g taxi dispatch, but may still be interesting for other applications.

The level of usefulness of the location information held by the GMS depends both on the configuration of the telecom system domain and the requirements of the application using it. We shall not discuss this further but concentrate rather on how to provide the necessary operations which can be invoked by the applications to obtain the required information from the GMS.

We introduce a new computational object called **Informer** which has the following operations that can be invoked from the applications:

```
GetNAP( in Userid, out NAPid, out Status)

GetGroupNAP( in GroupId, out ListOfNAP, out Status)

GetTerminal( in Userid, out Terminalid, out Status)

GetGroupTerminal( in GroupId, out ListOfTerm, out Status)
```

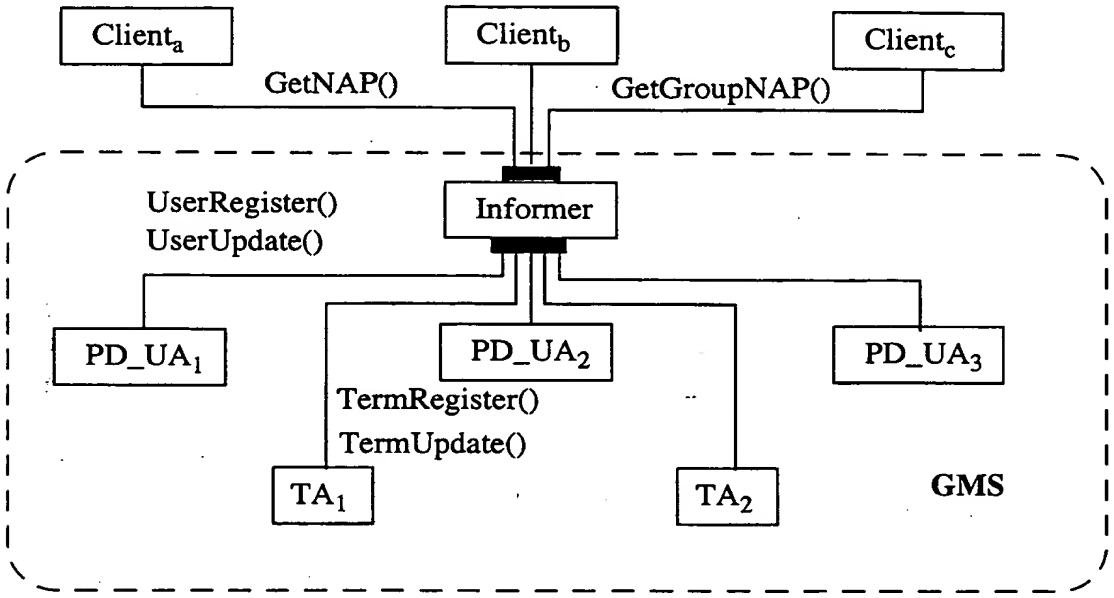


Figure 9.24 The Informer object provides location service to the applications

In order to obtain the location information, the **Informer** object needs assistance from both the **PD_UAs** and the **TAs**. All the users and terminals whose location should be available to the applications must be registered by the **Informer** object. Every time the user is registered with the **User_Registration** object the **PD_UA** has to notice the **Informer** object. This can be set up at subscription. The same applies for the terminal. Every time the **TA** update the **NAPid** saved in the **Terminal_Data** (see Chapter 5), the **TA** must notice the **Informer** object. This may also be configured at subscription time.

The **Informer** object should have the following additional operations:

```
UserRegister(in Userid, in PD_UAref, out Status)

TerminalRegister(in Userid, in TAref, out Status)

UserUpdate(in Userid, in Terminalid, out Status)

TermUpdate(in Userid, in NAPid, out Status)
```

As shown in Figure 9.24, the first group of operations we specified is used by client objects in the application in order to get location information while the second group is used by internal object of the GMS to supply information to the **Informer** object.

9.8 CLASSIFICATION OF APPLICATIONS ACCORDING TO THEIR MOBILITY AWARENESS

Before leaving the topic of application management, let us classify the applications according to their mobility awareness. Such a classification is relevant in the sense that the degree of mobility awareness reflects also the degree of dependency of the application on the GMS.

1. **Mobility-unaware** applications can be either incoming applications or outgoing applications which are not main applications or registration applications. These applications need no adaptation to become available to mobile users because they never interact directly with the GMS objects. When an application object residing in the telecom system domain interact with another residing in the user domain, their operations are intercepted and converted by the proxy object (see Chapter 8). They are then conveyed and delivered correctly to the addressed object. The application objects has neither knowledge about the proxy objects nor the conversion and conveyance of operations performed by the GMS. The application designer needs not be aware of the GMS when building such applications, although they use the functionality offered by the GMS as described above.

2. **Mobility-aware** applications can be main applications, registration applications or other management applications. These applications needs to use services explicitly from the GMS to accomplish their mission, i.e. the application designer must have knowledge about the structure and capabilities of the GMS.

3. **Mobility-based** applications are active client of the GMS and use the services offered by the GMS through the `Informer` object.

9.9 CONCLUSION

In this chapter the following existing objects of the GMS are modified and equipped with additional functions and operations:

- `User_Registration`
- `PD_UA`
- `TA`

The following new objects are added to the GMS:

- `SSM` (Service Session Manager)
- `PD_USM` (Provider Domain User Session Manager)
- `TD_USM` (Terminal Domain User Session Manager)
- `SF` (Service Factory)
- `Locator`

- `Informer`

10

Example of implementation

10.1 INTRODUCTION

In this chapter, we shall consider briefly the implementation and simulation at the Center for Technology at Kjeller, UNIK which are intended not only to test the concepts proposed in this thesis concerning the support of mobility in an ODP/TINA system but also to acquire practical experience with the design and implementation of distributed telecommunication applications in general. Before proceeding with the description of the simulation environment, a summary of the model for mobility support presented in this thesis will be given.

10.2 SUMMARY OF THE PROPOSED MODEL FOR MOBILITY SUPPORT

10.2.1 Model of a Functional Architecture supporting mobility

The model in Figure 10.1 depicts a situation where a user is using an application X at a terminal. The whole system is partitioned into three administrative domains: user domain, terminal domain and telecom system domain.

The user domain consists of two layers: security layer and application layer. Note that technologically the user domain can either rely on the terminal domain or run on a separate platform having communication capability with the terminal such as with two terminals connected in a tandem. The interactions between the user domain and the telecom system domain are logical rather physical since real interactions always go through the terminal domain and never directly.

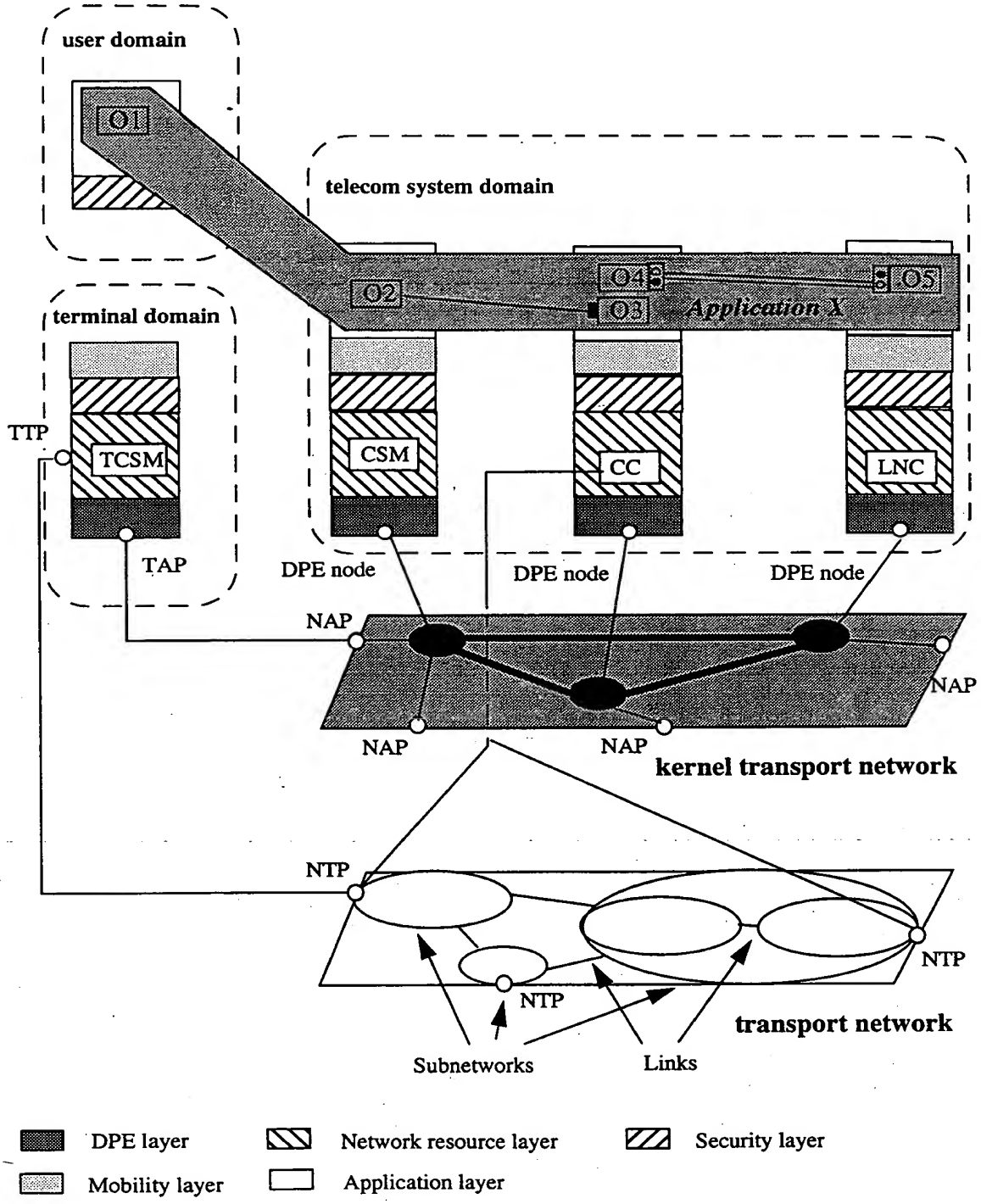


Figure 10.1 Model of a Functional Separation Architecture supporting mobility

The telecom system domain is shown with three DPE nodes. At each node, there are five layers: DPE layer, network resource layer, security, mobility layer and application layer. Recall that the layer separation here is not hierarchical but functional. Objects in one layer can communicates locally with objects in the same layer but also with objects in other layers (see Chapter 4).

The DPE layer uses the kernel transport network (kTN) for the end-to-end information transfer between its DPE nodes (also called computing nodes) [TIN94a], [TIN94b]. Operations between computational objects are also conveyed on the kernel transport network. The kernel transport network can be implemented using any standard telecommunication network such as packet-switched IP networks, ISDN, SS7 networks, LAN networks, connection-less or connection oriented networks. At the boundaries of the kernel transport network, there are multiple Network Access Points (NAP) allowing connections to be made onto the network.

The network resource layer comprising objects such as CSM (Communication Session Manager), CC (Connection Coordinator), LNC (Layer Network Coordinator), etc. uses a transport network to establish stream flows between computational objects [TIN95b]. Figure 10.1 shows a transport network consisting of a layered network containing several subnetworks. At the boundaries of the transport network, there are multiple Network Termination Point (NTP). The CC object is responsible for establishing connections between Network Termination Points.

In the terminal domain, there are four layers: DPE layer, network resource layer, mobility layer and security layer. A Terminal Access Point (TAP) is connected to a NAP of the telecom system domain as shown in the figure. Note that the connection can be wireline or wireless. A terminal may of course have several TAPs. In the network resource layer of the terminal, a Terminal Termination Point (TTP) is connected to a Network Termination Point of the telecom system domain.

The application X consists of the objects O1, O2, O3, O4 and O5 and spans two domains: user domain and telecom system domain. Application objects can interact with all objects in other layers.

10.2.2 Components of the GMS

As proposed earlier, the mobility layer is realized by the Generic Mobility System (GMS). The computational objects contained in the GMS are shown in Figure 10.2. Note that the grouping of objects according to their functions is only to make the figure easy to understand.

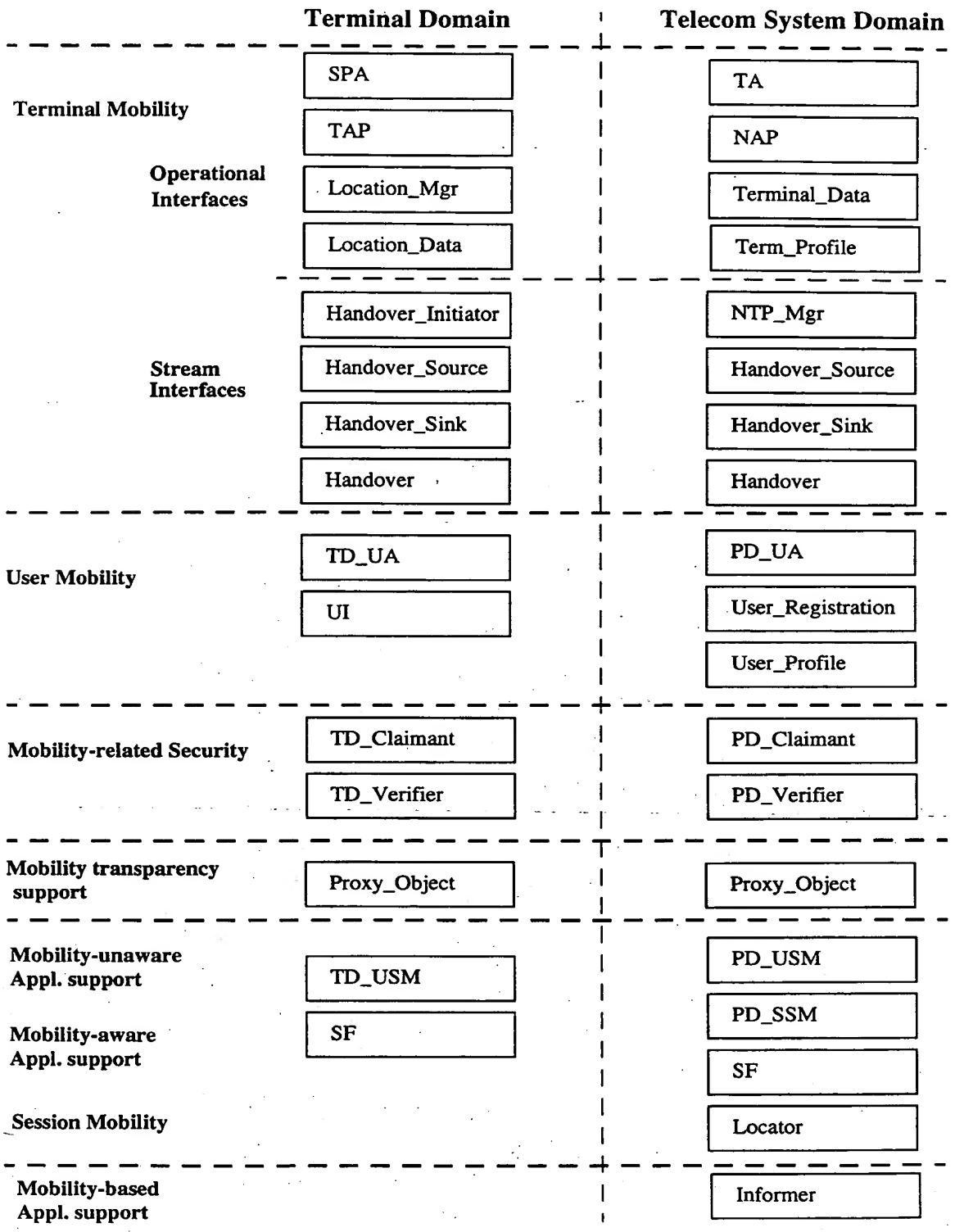


Figure 10.2 The components of the GMS

10.3 IMPLEMENTATION OF A SIMULATOR

In order to test and verify the model of the mobility support for a TINA/ODP telecommunication system proposed in this thesis, a simulated environment is suggested and attempts of construction have been initiated first in the framework of the SANDRA project and currently in the framework of TELEREP, the TELEcom Research Program at UNIK, the Center for Technology at Kjeller [Spi96].

A system consisting of the telecom system domain, terminal domain and user domain will be simulated in a computing environment which consists of a set of computers running SunOs or Solaris operating system and communicating with one another using TCP/IP on Ethernet. The TCP/IP-based Local Area Network is used both as kernel transport network and as transport network for streams. As application, Universal Personal Telecommunications (UPT) [CCI92c] [ITU94] [Wal93] [Soe93] [Sun93] will be designed and implemented in a TINA/ODP-compliant manner in order to test an advanced application. An ORB-implementation (Object Request Broker) [Objа], ORBeline, from PostModern Computing Technologies, Inc. is used as TINA-DPE [TIN94a], [TIN95d], [TIN94b].

The research work was organised as four Master thesis:

1. Design and implementation of Incoming Call and Outgoing Call
2. Design and implementation of Registration and Profile Service of UPT
3. Design and implementation of a simulated TINA telecommunication system for the testing of UPT
4. Design and implementation of persistent storage for Orbeline

At the time this thesis is written, the implementation is only partially accomplished. The software for some of the UPT application objects and some of the GMS objects is still to be written. This is a straightforward but laborious task. Below we shall give an overview of the design chosen for the simulator.

10.3.1 Specifications of the simulated system at UNIK

A simplified picture of the simulated system is shown in Figure 10.3. The system consists of a set of users U_1, U_2, U_3, etc., a set of terminals T_1, T_2, T_3, etc. and a set of NAPs (Network Access Point) NAP_a, NAP_b, NAP_c, etc. Some of the users are also owner of the terminals, some are not. In order to simplify, there is only one type of terminal and one type of NAP.

The simulated system will support continuous terminal mobility, personal mobility and session mobility.

As shown in Figure 10.3, a terminal T_c initially registered at NAP_2 can be moved and registered at NAP_1 without losing communications with the telecom system domain, i.e if there is a call going on while the terminal is moving, it will not be disrupted. Two terminals such as T_f and T_g can be attached to the same NAP_6. Con-

versely, NAP_7 is not connected to any terminal. Terminal T_d is not attached to any NAP. This simulates the cases where terminal T_d is out of coverage, switched off or defect.

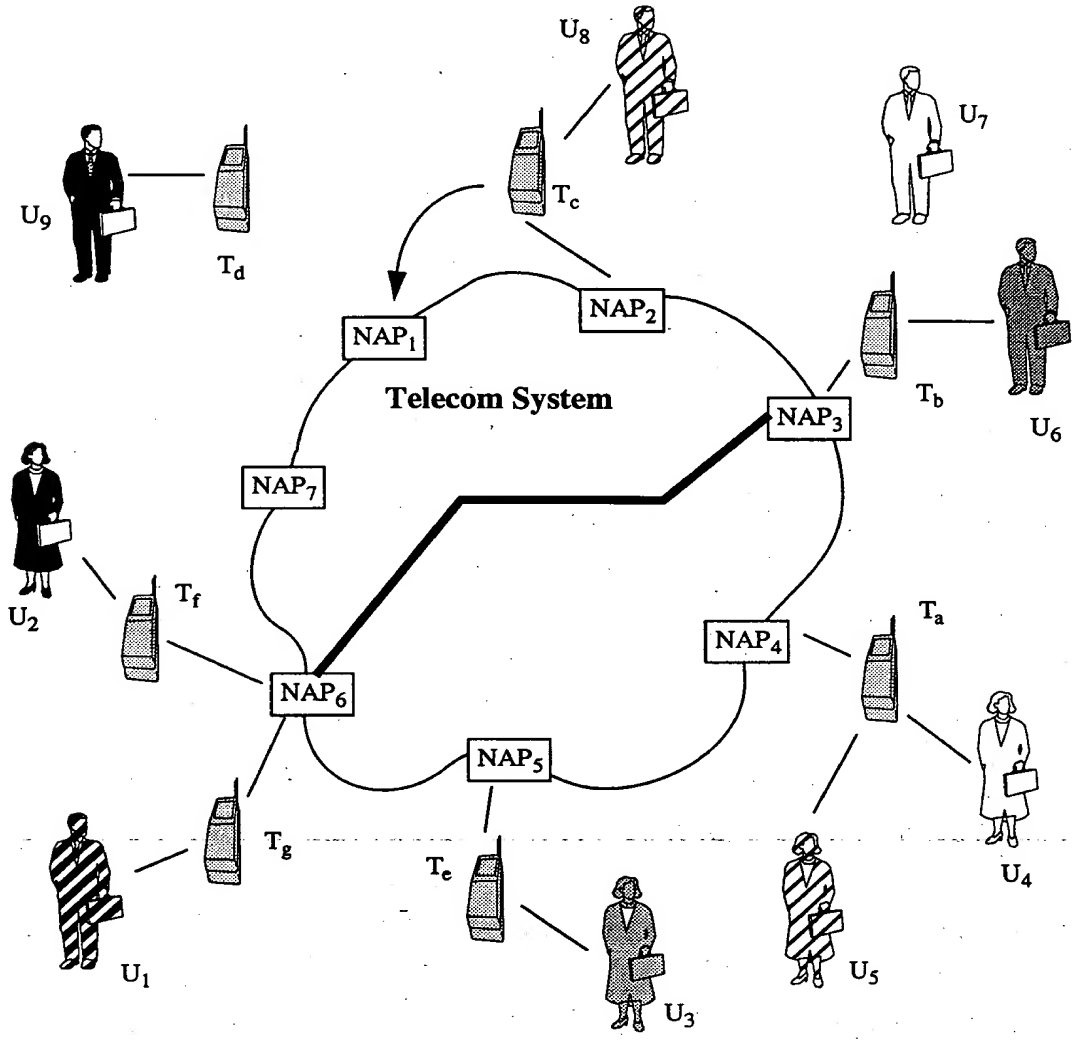


Figure 10.3 The simulated system at UNIK

Concerning user mobility a user can register at any terminal to receive or make calls. For instance, user U_6 is registered at terminal T_b which is attached to NAP_3 and is hence entitled to make or receive calls from any other user. A call between user U_6 and user U_1 is shown in Figure 10.3. Several users such as U_5 and U_4 can be regis-

tered at the same terminal T_a. User U_7 on the other hand, does not want to be disturbed and hence does not register himself at any terminal. For each terminal, there is

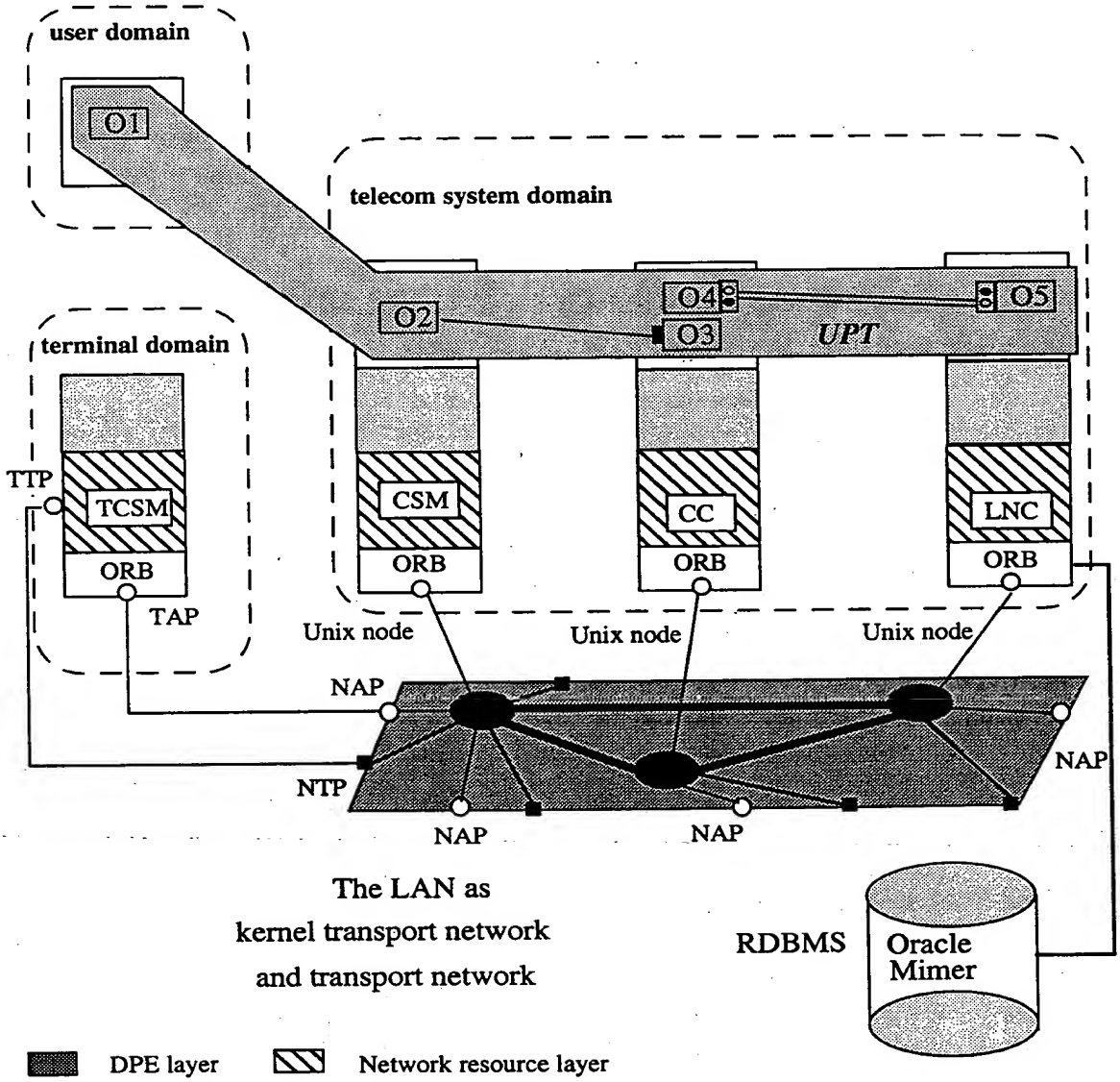


Figure 10.4 A Functional view of the simulated environment at Unik

always registered a default user who is the terminal owner. A user can also suspend an ongoing call at one terminal, move to another terminal and resume the call there.

Some of the personalisation features defined by UPT [CCI92c], [ITU94] such as incoming call screening, outgoing call screening, call forwarding, routing by time table,

etc. will also be available in the simulated system. These objects have not yet been written.

10.3.2 The simulated environment in a functional view

In the simulated system the telecommunication network is mapped to the telecom system domain while each terminal is mapped to one instance of the terminal domain and each user to one instance of the user domain. Note that the user domain may not contain only the user himself but also other objects representing software processes or hardware devices. We shall now consider successively the telecom system domain, the terminal domain and the UPT application.

10.4 THE SIMULATED TELECOM SYSTEM DOMAIN

The simulated telecom system domain will have four layers: DPE layer, network resource layer, mobility layer and application layer. The security layer has been omitted for the time being because of its complexity.

10.4.1 The DPE layer

In the DPE layer an ORB-implementation Orbeline [Pos] is used as DPE and installed in all the computers of the domain. Each of these computers is thereafter referred as a computing node (OMG terminology) or DPE node (TINA terminology). Orbeline supports access and location transparencies for operational interactions between computational objects located in the telecom system domain. A client object can request an operation on any other object by specifying the object reference and the desired operation. Orbeline is responsible for all the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request [Objb].

A virtual boundary is created at the DPE layer for the telecom system domain. This boundary is represented by a set of simulated NAP (see paragraph 10.4.3). Any operation crossing the domain boundary between the telecom system domain and the Terminal or user domain has to be conveyed through a NAP.

Interactions in form of stream flows are not supported by Orbeline (By the time the work was started, the stream concept was not defined in the ORB specification. The concept was introduced by TINA). Stream flows will, however, not be implemented in the DPE layer but in the network resource layer.

Although Orbeline maintains the persistence of object references, its does not provide sufficient storage for the complete state of an object. External database system must connected to Orbeline and used for the storage of the necessary data.

This is actually the goal of the Master thesis no. 4 mentioned above. To use an Object DBMS (Object Database Management System) is straight forward because ODBMSs have been developed to store objects. However, the ODBMS is an unmatured

concept and there is not yet consensus about the criteria for an ODBMS. Since relational DBMS has been here for a long time and the products are more mature, it is decided to integrate a relational DBMS into the DPE platform. We first used an Oracle DBMS but change to Mimer DBMS from Sysdeco because of better customer support and service. The implementation of persistent storage has been completed. We shall only consider this subject briefly and the reader interested in more details is referred to [Do96a], [Tra97].

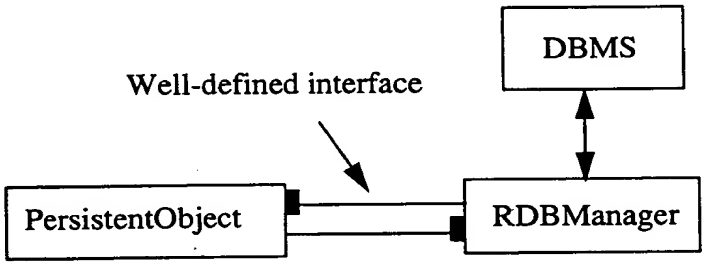


Figure 10.5 A base type for all the persistent objects

Briefly, the design is based on the fact that persistence is a characteristic which can also be inherited. A **PersistentObject** type is defined and all persistent objects will inherit from this base type. As shown in Figure 10.5, the **PersistentObject** type has well-defined interface with the **RDBManager** (Relational Database Manager) object which handles interactions with the DBMS. High level of flexibility is achieved in this way since introduction of new DBMS does not have any impact on the **PersistentObject** type. What is required is a new **RDBManager** having a new interface with the new DBMS but an unmodified interface with the **PersistentObject** type. This interface consists of operations such as `get()`, `put()`, etc. and the information which is passed across the interface is the data to be stored in or retrieve from the database in the database format.

10.4.2 The Network resource layer

Since the transport network used in our environment is a Local Area Network, there is no need for all the computational objects specified in the TINA-C Connection Management Architecture [TIN95b] such as the **Layer Network Coordinator** (LNC), the **Connection Performer** (CP), **Connection Management Configurator** (CMC), etc. which are intended to support the establishment and release of connections on a large and complex transport network consisting of several layered networks which may consist again of several subnetworks. Only the **CSM** (Communication Session Manager) defined by TINA being the object providing the service of binding stream interfaces of object is required in our simulated environment. As shown in Figure 10.6, an arbitrary **Client object** can request the **CSM** to establish a stream between two objects A and B.

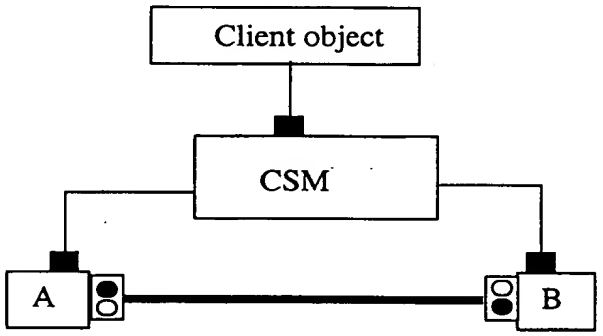


Figure 10.6 Objects involved in the binding of stream interfaces

The simulated `CSM` will have the same interface toward the `Client` object as the real one but instead of invoking operations on the `CC` (Connection Coordinator), `TCSM` (Terminal Communication Session Manager), etc. to establish or release a stream, the simulated `CSM` invokes operations directly on the objects `A` and `B`.

The objects A and B need both to encapsulate the functionality necessary to realise streams. A base type called `Streamtype` has been designed, and all objects having stream interfaces will inherit from `Streamtype`.

The stream flow between objects are implemented using the Unix Transport Interface [SUN90]. The implementation of the CSM and the stream is the goal of the Master thesis no. 3 mentioned above. This work has been completed. We shall not study this subject further here but refer the interested reader to [Do96a], [Ngu97].

In order to simplify the situation, we let the transport network also span over the terminal domain. There is therefore no border in the transport network between the two domains. This means that streams can be established with objects residing in the terminal domain by the same mechanism as described above. There is no need for the `NTP_Mgr` object intended to manage the Network Termination Point (NTPs) (See paragraph 5.3).

There is no need for the **Handover** objects because in our environment, objects will never move relatively to the transport network.

10.4.3 The mobility Layer

The mobility layer is spanning over the telecom system domain and terminal domain and is realised by an instance of the GMS. We shall now study those objects of this GMS instance which belong to the telecom system domain.

The simulated NAP

The kernel transport network is realised in the simulated system by the local area network but the real boundary and the real access points of the local network are not used. Instead, a virtual boundary with virtual access points are defined. The kernel transport network relies on the local network but is smaller. The simulated `NAP` object represents this virtual access point. A set of NAPs will be instantiated in the simulated system.

In order to support continuous terminal mobility each `NAP` must have a coverage area and the coverage areas of adjacent `NAP` overlap one another. The NAP coverage area is however only virtual in our environment. As shown in Figure 10.7, we define a NAP coverage area in such a way that it has only six adjacent areas and the NAP area overlaps only two by two, as indicated by the shaded regions, i.e the intersection of three or more NAPs is empty.

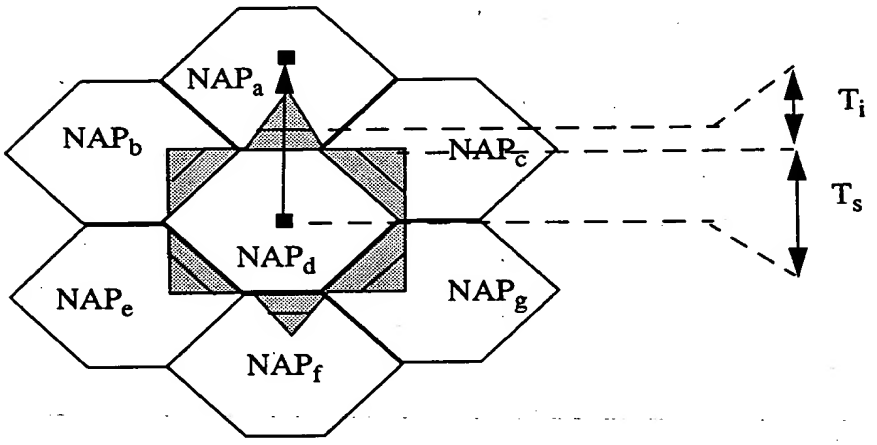


Figure 10.7 Movements of the terminal across the NAP coverage area

The movements of a terminal must also be simplified. A terminal located in one NAP is allowed either to move inside the NAP area or to migrate toward one of the six neighbouring NAP areas. Once the migration is started, it can not be reversed before completion, i.e the terminal must arrive at the new NAP area before any other move can be allowed. The migration is not defined in terms of distance but in terms of time. Figure 10.7 shows a terminal migrating from NAP_d to NAP_a. After a time period T_s, the terminal is considered as entering the intersection area. Here, registration at the new NAP_a is done but communications still go through NAP_d. After a time T_i, the terminal reaches the frontier of change and communications are switched over to

NAP_a. Note that the initial position of the terminal before the migration is always assumed to be the centre of the NAP area.

The simulated TA (Terminal Agent)

The simulated `TA` has the same functions has the real `TA` (see Chapter 5). In the telecom system domain, there will be one `TA` for each terminal.

The simulated PD_UA (Provider Domain User Agent)

The simulated `PD_UA` has the same functions has the real `PD_UA` (see Chapter 6). In the telecom system domain, one `PD_UA` will be instantiated for each user.

The following objects have the same functionality as the real ones:

- `Term_Profile` (see Chapter 5)
- `User_Profile` (see Chapter 6)
- `Terminal_Data` (see Chapter 6)
- `User_registration` (see Chapter 6)
- `Proxy_Object` (see Chapter 8)
- `PD_USM` (see Chapter 9)
- `PD_SSM` (see Chapter 9)
- `SF` (see Chapter 9)
- `Locator` (see Chapter 9)

The following objects have only reduced functionality:

- `PD_Claimant` (almost empty)
- `PD_Verifier` (almost empty)

The object `Informer` is omitted since application requiring its services is not implemented.

10.5 THE SIMULATED TERMINAL DOMAIN

No hardware device will be used as terminal in the simulated system. Instead, a terminal will be simulated by an X client application as shown in Figure 10.8. The simulated terminal consists of the simulated versions of the objects defined for the terminal domain such as `TAP`, `Location_Mgr`, `Location_Data`, etc. plus additional objects for graphic display on an X server. Several instances of simulated terminal can be displayed on the same X server. These instances will be running on the same computing nodes (Unix machines) as the Telecom_System domain. However, as we will see later that virtual boundaries will be created to separate the terminal domains from the Telecom_System domain.

We proceed now with the composition of the terminal domain by considering each functional layer in details.

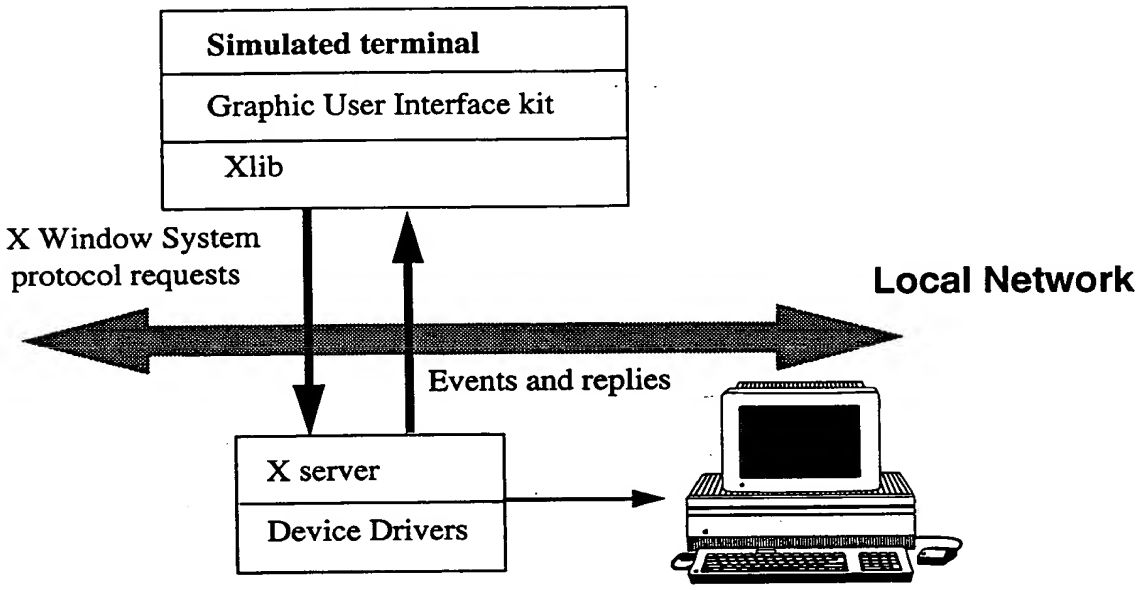


Figure 10.8 Simulating the terminal as an X application

10.5.1 The DPE layer

The terminal domain is actual running on the same DPE and using the same kernel transport network as the telecom system domain. A virtual boundary is created at the DPE layer for the terminal domain. This boundary is represented by one simulated `TAP` (see below). Any operation crossing the domain boundary has to be conveyed through this `TAP`.

10.5.2 The Network resource layer

In order to simplify the system, we let the transport network also cover the terminal domain. The same mechanisms to support streams in the telecom system domain can therefore be used in the terminal domain. Consequently, there is no need for `Handover` objects and the `TCSM` (Terminal Communication Session Manager) objects in the simulated terminal domain.

10.5.3 The mobility Layer

The simulated objects having the same functionality as the real ones are:

- `TD_UA`
- `SPA`

- `TAP`
- `Location_Mgr`
- `Location_Data`
- `Proxy_object`
- `TD_USM`
- `UI` (User Interface)
- `SF` (Service Factory)

The simulated objects having only simplified functionality are:

- `TD_Claimant`
- `TD_verifier`

Simulation of the movements of the terminal

Each instance of the terminal domain once installed on a computing node in the simulated system will remain there until an eventual reconfiguration. The mobility of the terminal will be virtually created. As specified in paragraph 6.4.3, a wireless terminal is made aware of its motion through the received location information, i.e the NAPid which is periodically broadcast by the NAP. If the new NAPid is different from the previous one, the terminal knows that its has moved from one NAP area to another and, depending on both the context (terminal state) and the received information, it will initiate appropriate procedures such as location registration, location updating, etc. It is therefore possible to make the simulated terminal believe that it has moved by feeding new NAPid into it. The computational objects in the terminal domain will therefore behave in the same way as real terminals.

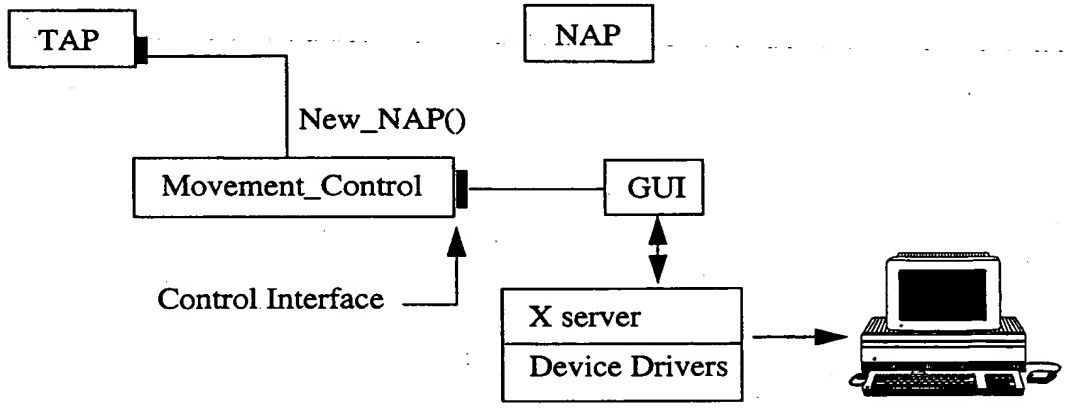


Figure 10.9 Movement_Control is the object responsible for the movement simulation

As specified in paragraph 6.4.3, the **TAP** is the object in the terminal domain that receives the NAPid broadcast by the **NAP**. It needs to be modified in the simulated system in order to receive new NAPids. The simulated **TAP** is equipped with an operation called `New_NAP()` which is used for the transfer of new NAPid. When the new NAPid is Nil, it means that the terminal has moved out of coverage and the TAP will cease to convey operations no matter whether they are requested from the terminal domain or the telecom system domain.

The simulated **NAP** does not broadcast its NAPid. Instead, as shown in Figure 10.9, an object called **Movement_Control** is introduced to assume this responsibility by requesting the operation `New_NAP()` on the **TAP**. Only one instance of the **Movement_Control** object is required in the simulated environment to control the movements of all the terminals. Its has an interface called `Control` which allows the initiation of terminal movement. This interface is accessible via a window running on an X server. All possible movements between NAPs are possible including moving in and out of coverage (see above)

The simulation of a terminal move from one NAP coverage area to another is realized as follows. The **Movement_Control** object starts an internal timer which will time out after Ts + Ti seconds, where Ts is the time used to reach the overlap area and Ti is the time used to reach the switchover point. The **Movement_Control** object will then invoke the `New_NAP()` operation on the **TAP** and deliver the identifier of the new NAP. The **TAP** will transfer the received NAPid to the **Location_Mgr**. The **Location_Mgr** compares the new NAPid with the one saved in the **Location_Data**. If they are identical, nothing will be done. If they are different and are both different from Nil, the **Location_Mgr** will initiate a location updating procedure (See paragraph 5.2.4). If the old NAPid is Nil, a registration procedure is performed. If the new NAPid is Nil, the terminal will remain silent until it receives a NAPid different from Nil.

In addition to controlling the movements of the terminal, the **Movement_Control** object also offers the capability to switch off the terminal. Upon receipt of a `switch_off` request, the **Movement_Object** will deactivate all the objects in the terminal domain.

10.6 THE UPT APPLICATION

The UPT application consists of four modules:

- Outcall is an outgoing application permitting a user to make outgoing access. Features such as roaming restriction and outgoing screening are also included. This application is mobility-unaware.
- Incall is an incoming application which delivers calls to the users. Features such as incoming screening and roaming restrictions are included. This application is also mobility-unaware
- Registration is an outgoing application which permits a user to register his lo-

cation in order to make or receive calls. Since it interacts directly with the GMS, this application is mobility-aware.

• Profile management is an outgoing application which enable the user to tailor the UPT according to his preferences. Since it interacts directly with the GMS, this application is mobility-aware.

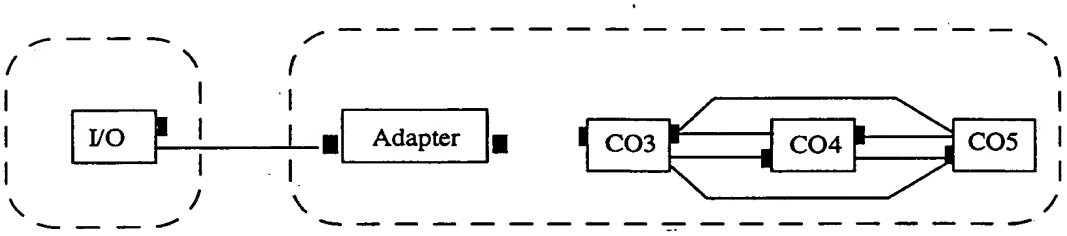


Figure 10.10 Simple representation of a UPTmodule

Each of the application module can be represented as shown in Figure 10.10. Since there is only one type of terminal in the simulated system, only one type of I/O object and one type of `Adapter` object are required for each module. Whenever an instance of a application module is instantiated for a user, the GMS will assist in creating the I/O object in the correct virtual terminal. In our simulator, two incall modules can be instantiated at the same time on the same terminal and, consequently, the busy state does not exist with our terminal.

At the time of writing, the Master thesis where the UPT module is designed an implemented has not been completed.

11

Conclusions

In this thesis, we propose and present a Generic Mobility System which supports mobility transparency in the telecommunications system. The GMS can be customized and configured to contain the type of mobility required and to fit into any type of telecommunications system, private or public, wireless or wireline, wide-area or local-area. We conclude this work with a critical review of the claims we stated in paragraph 1.3. Finally, we suggest and discuss some areas for future research.

11.1 CRITICAL REVIEW OF CLAIMS

Below we summarize the contribution of this work by critically reviewing the fulfilment of our claims as presented in Chapter 1.

Claim 1: We demonstrate that the access and location transparencies cannot alone support the support of terminal mobility and personal mobility.

Chapter 4 shows that in an ideal model where all objects are accommodated on one DPE, terminal mobility and personal mobility are seamlessly supported only assuming that access and location transparencies are supported. However, in a real system where interoperability between the terminal DPE and the telecom system DPE is not guaranteed at all times, terminal mobility is no longer automatically supported. The user (person, machine or software application) may be a software process or represented by a software process running on a DPE. For the same reasons as for the terminal mobility, continuous interoperability with DPEs of the Telecom system cannot be guaranteed. As claimed, we have shown that personal mobility is no longer automatically supported.

Claim 2: We propose a Functional Separation architecture supporting mobility.

In Chapter 4 we propose a Functional Separation architecture where the mobility functions are separated both from the network resource layer and the application layer. Greater flexibility is achieved by this architecture since each layer in the architecture can be developed independently. The proposed architecture is used throughout

the thesis and its implementation is proven to be possible in chapter 10 by the implementation of a simulated telecommunications system.

Claim 3: We propose a Generic Mobility System realising the Mobility layer of this architecture.

In order to realise the mobility layer, a Generic Mobility System (GMS) is outlined in Chapter 4. The functions supported by the GMS are successively studied and designed in Chapter 5 through 9. The GMS is gradually populated with computational objects as different aspects of mobility are identified and studied. The GMS is customisable in the sense that its components can be selected, and possibly, modified to suit the requirements of new types of telecommunications systems. The GMS can also be implemented as middleware, e.g as off-the-shelf software for a distributed systems using CORBA.

Claim 4: We develop and design the required mechanisms for terminal mobility support.

Chapter 5 is dedicated to the identification and specification of the functions required for supporting terminal mobility. There are two types of interactions between the terminal and the telecom system domain, namely operations and stream flows. For conveyance of operations, the kernel transport network is divided into a mobile part and a fixed part. The border between the two parts consists of a set of Network Access Points (NAP) which constitute the gateways between the two parts. Four computational objects **TA**, **NAP**, **TAP** and **SPA** are introduced in order to ensure the conveyance of any operation between objects in the terminal domain and objects in the telecom system domain. For stream flows, the **NTP_Manager** object is introduced to control and supervise the Network Termination Points (NTPs) which are the gateways of the transport network. The **NTP_Manager** object together with the **Handover** objects ensure the continuity of the stream flows when the terminal is moving. Terminal mobility is hence supported.

Claim 5: We develop and design the required mechanism for user mobility support.

In Chapter 6 a **PD_UA** object is introduced to represent the user in the telecom system domain and a **TD_UA** object is introduced in the terminal domain to support user functions locally. At each terminal the user is using, there is created an instance of the **TD_UA** object. By keeping track of where the **TD_UA**s are located the **PD_UA** object ensures that operations can be sent and delivered correctly to objects residing in the user domain and the telecom system domain, while the user are moving. For stream support it turns out that no additional object are required. User mobility is hence supported in our model.

Claim 6: We develop the required mechanisms for the mobility-related security functions.

Chapter 6 treats the security problems which are aggravated by mobility. Whenever the terminal domain tries to re-establish contact with the telecom system domain, the authentication and access control procedures must be executed to protect both the tele-

com system and the terminal owner. These procedures involve both terminal domain objects such as `TA`, `Term_Profile`, etc. and security objects such as `Claimant`, `Verifier` and `ADF`. Corresponding procedures are also studied for the user and especially, the access control for the use of the terminal intended to protect the terminal owner against unauthorised use of his terminal.

Claim 7: We introduce an approach by which mobility is made transparent to applications.

Three alternatives to integrate the GMS into the system are studied in Chapter 8, namely the In-Line, the Broker and the Proxy alternatives. The proxy alternative is chosen since it provides the highest level of transparency. By the use of proxy objects, other objects believe that they interact directly with their counterparts and are unaware that interactions are actually redirected to proxy objects which pass the request to the GMS. The operation requests are then conveyed inside the GMS across domain boundaries and delivered correctly to the addressed object.

Claim 8: We propose guidelines regarding how applications are integrated into the telecommunications system.

Chapter 9 treats application as entire and single units. Applications are classified into two types according to the initiator. Relative to one user we define outgoing and incoming applications. Outgoing applications are initiated by the user while incoming application by other users or processes in the telecom system domain. The relations between the applications and the GMS are described. Guidelines concerning how to integrate applications are also considered in details. It is shown that applications such as telephone, videophone, information services and word processing can be made available to mobile users without any modifications. They are mobility-unaware. To work properly, however, the GMS needs the assistance of the mobility-aware applications such as the registration application. Otherwise, the application objects in the user or terminal domain cannot be located.

Claim 9: We develop the required mechanism for session mobility support

The session mobility support is handled in Chapter 9. The user can move from one terminal to another without having to terminate a session and re-start it from the beginning. With the assistance of the GMS objects such as `PD_UA`, `SSM`, `TD_USM`, etc. the user can suspend the session and then resume it at another terminal.

Claim 10: We introduce service interface for mobility-based application

For mobility-based application, i.e applications that use explicitly the mobility-related information, a service interface is introduced and described in Chapter 9. The object offering this interface is the `Informer`.

Claim 11: We show that the GMS can be implemented.

In order to verify that the proposed Functional Separation Architecture is applicable and that the GMS can be implemented, an example of implementation is describer in Chapter 10. Although the implementation is not completed, we can, nevertheless, at

this stage conclude that it is possible to implement the GMS on a CORBA-type platform. The work that remains to be done is mainly laborious programming of the required objects.

11.2 AREAS OF FURTHER WORK

11.2.1 Replication of processing objects and data objects

In this thesis the engineering viewpoint is considered only briefly. In order to function the GMS requires only the location transparency and the access transparency in the DPE of the telecom system domain. However, in order to function efficiently, the GMS will also require the replication transparency and a replication strategy.

In the telecom system domain, computational objects (CO) can be divided into three types: processing CO, data CO and combined processing and data CO. To make the discussion simple, we consider only the first two types. The processing COs, which are interesting in our current discussion, are the agent objects, for instance the `User_Agent` (UA) object. The data COs are the profile objects, for instance the `User_Profile` (UP) object.

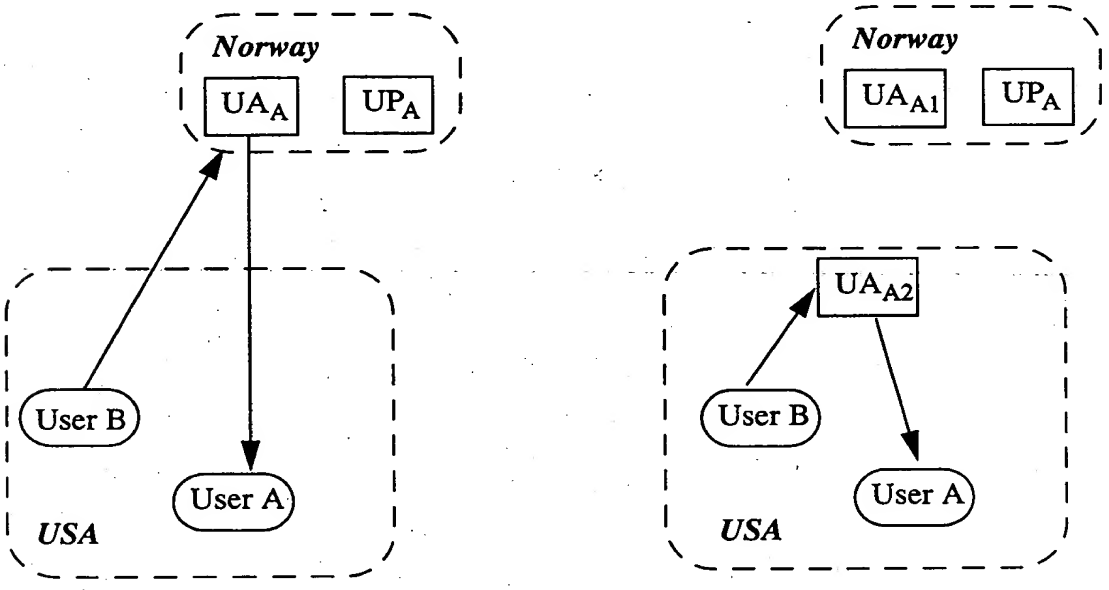


Figure 11.1 Examples of the replication of processing objects

In the thesis we assume that for each user there is only one instance of the UA and that this instance has only one replica. The same applies for the profile, there is just

one instance of the UP and also only one replica. The GMS is of course able to function properly in these conditions but not necessarily efficiently as shown in the case depicted by Figure 11.1.

In case a, User A subscribes to the telecommunications services in Norway and is allocated a UA_A and UP_A in Norway. User A is currently in the USA. A User B resident in the USA wanting to contact User A, has to get in contact with the UA_A in Norway which resend the request back to User A in the USA. There is here an overhead in the conveyance of the operation between A and B. As shown in case b, this overhead can be removed by having a replica of UA_{A2} in the USA. However, this requires brokers which know where to look for the UA object.

Similarly, the data object UP can also be replicated. Unfortunately, replication may improve the efficiency but introduces also several problems which need to be solved before replication can be used. First, how should the replicated processing and data objects be related to each other with regard to achieving consistency and reflecting the mobility of the user? Should there be a common replication strategy applied for all the users or a separate one for each user? Second, how can the registration of a user which is at the application layer be associated with the replicated objects which reside in the DPE layer? Third, how can consistency be preserved or how much inconsistent should the system tolerate? These subjects are interesting and deserve to be studied further.

11.2.2 Migration of objects

Another solution to the overhead problem mentioned above is to use migration of objects. Instead of having replica of the UA and UP objects, it may be wiser to move them along with the user. The first problem is that migration is currently used only in fault situations and is thus initiated and controlled by the system and not by the application. A mechanism allowing the application to control the migration of its objects must be developed. A strategy for the migration of the user's objects in correlation with the user's movement must also be established. Such a strategy can be common for all the users or specific for each users.

11.2.3 Mobile Agent

Another solution to the same problem is perhaps to use the concept of Mobile Agent. A mobile agent is an object that has the ability to move and to start executing autonomously at its new location[OMG96]. Instead that the migration of the agent object is initiated and controlled by an application as in the previous solution, the mobile agent may have the capability from one environment (machine) to another to decide about its own migration. Mobile agent is an emerging technology which attracts interest from the fields of distributed system, information retrieval, electronic commerce and artificial intelligence.

11.2.4 Intelligent Agent

If the intelligence of the mobile agents is developed further then they may become intelligent agents. The intelligent agent is then not handling only the user mobility problems but is also capable to decide and deal with other problems such as man-machine interface, personalisation, preferences, etc. The application of intelligent agents to telecommunications problems is new and merits further study.

11.2.5 Internet, Java and Network Computer

Internet brings along new paradigms such as Java and Network Computer (NC). Java is an object-oriented programming language and interpreted language. Rather than producing machine-specific instructions, the Java compiler produces vendor-neutral bytecode. The Java runtime environment or virtual machine translates the bytecode into actual machine-specific instructions. The Java virtual machine is installed on the user's machine, either as part of a Web browser or as part of the underlying operating system. A Network Computer is a computer with minimal memory, disk storage and processor power designed to connect to a network, especially the Internet.

If we consider Network Computers as terminals and use as development and runtime environment a Java ORB it will be interesting to study how the GMS can be used to support application mobility, i.e enable the applications to move and follow the user. Interesting problems are:

- How can NCs be moved from one terminal and reconfigured on another terminal with different characteristics?
- How can the terminal domain objects defined in this thesis be implemented on an NC?
- How should we define the boundary between an NC terminal domain and the telecom system domain? Does the proxy concept defined in this thesis represent an appropriate interface between the NC and the GMS?

11.3 FINAL REMARKS

At the beginning of this work, we found the ODP/TINA concepts very complex, difficult to understand and confusing at several points. There are many ways to interpret the same notion and the same definition. For instance, how should the mapping between different viewpoints be done? In this thesis we did not attempt to do such a mapping. Instead we started with the enterprise viewpoint which forms the foundation for the design. Thereafter, the information viewpoint and the computational viewpoint were developed in parallel and iteratively. The system was in this way gradually populated with new objects until completion. The ideas, functions and features are passed from one viewpoint to another automatically without the need of any explicit mapping. We did however check and verify consistency between viewpoints.

Afterwards, through practical use, our opinion about ODP /TINA concepts started to change. The concepts become gradually reasonable and did alleviate our job in the design of our system. A lot of time and practice are, however, required to understand the concepts of ODP/TINA due to their level of abstraction and their vagueness. Literature containing guidelines for the design and implementation or examples of use are unfortunately rare and most of the models developed so far have been done by the TINA Core Team and by some of the auxiliary projects. Even if the mobility support proposed in this thesis is not adopted for real systems, we still hope that this thesis can serve as an example of using ODP/TINA concepts in the design and implementation of complex distributed systems.

References

[AB93] Jan A Audestad and Frank Bruaroey. Assessing the current evolution of personal mobility: Development of upt. Technical report, Telenor Research and Development, 1993.

[AJ92] A Jan Audestad and Boerge Jacobsen. Universal personal telecommunication and intelligent network architecture. *Telektronikk*, 2.92, 1992. Televerkets forskning Institutt.

[Aud92] Jan A Audestad. An introduction to intelligent networks. *Telektronikk*, (2.92), 1992.

[Aud96] Jan A Audestad. Introduction to distributed processing in telecommunications systems. Norwegian University of Science and Technology, Departement of Computer Systems and Telematics, February 1996. Lecture notes - Course 45356 Communication in distributed systems.

[Bel92] Bellcore. *The Bellcore OSCA Architecture*, issue 1 edition, October 1992.

[BN95] Ron Ben-Natan. *Corba - A guide to Common Object Request Broker Architecture*. Computing McGraw-Hill, 1995.

[Boo91] Grady Booch. *Object Oriented Design with applications*. The Benjamin/ Cummings Publishing Company, Inc., 1991.

[CCI92a] CCITT. *CCITT Recommendations on Intelligent Networks, Q.1200-1290*, 1992.

[CCI92b] CCITT. *Principles for a Telecommunications Management Network*, 1992. CCITT Recommendation M.3010.

[CCI92c] CCITT. *Principles of Universal Personal Telecommunication (UPT)*, 1992. Draft Recommendation F.850, version 4.

[Cro96] Jon Crowcroft. *Open distributed systems*. UCL Press Limited, 1996.

[Do96a] van Thanh Do, Jan A. Audestad. Experiences using corba to build telecommunications applications. In *Proceedings of the IFIP/ICCC International Conference on Information Network and Data Communication*, number ISBN 0 412 75750 8. Chapman & Hall, June 1996. Trondheim, Norway.

[Do96b] van Thanh Do, Jan A. Audestad. Making mobility transparent to the applications. In *Proceedings of the IEEE VTS 46th Vehicular Technology Conference*, number ISBN 0 7803 3157 5. IEEE VTS, IEEE, April 1996.

Atlanta, GA USA.

[Do96c] van Thanh Do, Jan A. Audestad. Mobility and tina. In *Proceedings of the TINA 96 Conference*, volume ISBN 0. TINA-C, September 1996. Heidelberg, Germany.

[Do96d] van Thanh Do, Jan A. Audestad. *Universal personal Telecommunication: Information Model.* UNIK, September 1996.

[Eri94] Ericsson. *UPT Phase 1 Service Function Specification*, July 1994. Revision 1.

[ETSa] ETSI. *NA:UPT: Service Requirements on protection of third parties.* Versin:1.0.0.

[ETSb] ETSI Radio Equipment and Systems (RES). *Digital European Cordless Telecommunications (DECT)reference document.*

[ETSc] ETSI Radio Equipment and Systems (RES). *Digital European Cordless Telecommunications (DECT) system description document.*

[ETS93] ETSI. *Network Aspects (NA); Universal Personal Telecommunication (UPT UPT Vocabulary)*, Sept 1993.

[ETS94a] ETSI. *European digital cellular telecommunications system (Phase 2); Mobile Application Part (MAP)specification GSM 09.02*, March 1994.

[ETS94b] ETSI. *European digital cellular telecommunications system (Phase 2); Organisation of subscriber data GSM 03.08*, May 1994.

[ETS94c] ETSI. *European digital cellular telecommunications system (Phase 2); Security related network functions GSM 03.20*, May 1994.

[ETS95] ETSI STC NA6. *Cordless Terminal Mobility (CTM) Functional Architecture and Procedure - Phase 1*, March 1995. Draft.

[EUR93] EURESCOM. *PROJECT P102, Universal Personal Telecommunication (UPT)*, July 1993. Final deliverable.

[GSM94a] ETSI. *European digital cellular telecommunications system (Phase 1); Mobile Application Part (MAP) Part 1:Generic GSM 09.02*, July 1994.

[GSM94b] ETSI TC-SMG. *European digital cellular telecommunications system phase 2; Location registration procedures*, september 1994.

[GSM94c] ETSI TC-SMG. *European digital cellular telecommunications system phase 2; Mobile Application Part (MAP)*, july 1994.

[Heg96] Hans Hegeman. *Support of User Mobility in the TINA Service Architecture.* TINA-C, June 1996.

[IB94] T. Imielinskii and B.R. Badrinath. Mobile wireless computing. *Communication of the ACM*, 37(10):10–28, October 1994.

[INA93] Bellcore. *INA Cycle 1 Framework Architecture*, issue 2 edition, April 1993.

[ISO92] ISO/IEC. *Information Technology - Open Systems Interconnection - securi-*

ty Frameworks in Open Systems - Part 3: Access Control, June 1992.

[ISO93a] ISO/IEC. *Information Technology - Open Systems Interconnection - security Frameworks in Open Systems - Part 1: Security Frameworks Overview*, August 1993.

[ISO93b] ISO/IEC. *Information Technology - Open Systems Interconnection - security Frameworks in Open Systems: Authentication Framework*, July 1993.

[IT94] Inc. IONA Technologies, SunSoft. *ORB Interoperability - ORB 2.0 RFP Submission.* Object Management Group, Inc., March 1994.

[ITUa] ITU-TS. *Basic Reference Model of Open Distributed Processing - Part 1 Overview and guide to use the Reference Model.* Rec.X901(ISO/IEC 10746-1).

[ITUb] ITU-TS. *Basic Reference Model of Open Distributed Processing - Part 2: Descriptive model.* Rec. X902(ISO/IEC 10746-2).

[ITUc] ITU-TS. *Basic Reference Model of Open Distributed Processing -Part 3: Prescriptive model.* Rec. X903 (ISO/IEC 10746-3).

[ITUd] ITU-TS. *Basic Reference Model of Open Distributed Processing -Part 4: Architectural Semanitcs.* Rec. X904 (ISO/IEC 10746-4).

[ITU94] ITU-TS. *Universal Personal Telecommunication (UPT) Service Description*, version 10 edition, January 1994. Draft recomendation F.851, version 10.

[Kat94] Randy H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, Vol 1(No 1), First Quarter 1994.

[Lin94] P.F. Linington. Rm-odp:the architecture. In Liz Raymond, Kerry & Armstrong, editor, *Open Distributed Processing experiences with distributed environments - Proceedings of the third IFIP TC6/WG 6.1 international conference on open distributed processing, 1994.* Chapman & Hall, 1994. Brisbane, Australia.

[MP92] Michel Mouly and Marie-Bernadette Pautet. *The GSM system for mobile communications.* Mouly & Pautet, 49, rue Louise Bruneau, F-91120 PALAISEAU - FRANCE, 1992.

[MSG94] Refik Molva, Didier Samfat, and Tsudik Gene. Authentication of mobile users. *IEEE Network*, Vol 8(2), March/April 1994.

[Ngu97] Dinh Quyen Nguyen. Design and implementation of a tina compliant environment for the testing of upt. Master's thesis, University of Oslo, Institutt for Informatikk, 1997.

[Obja] Object Management Group, Inc. *The Common Object Request Broker: Architecture and Specification.* Document No. 91.12.1, Revision 1.1, ISBN 0-471-58792-3.

[Objb] Object Management Group, Inc. *The Common Object Request Broker: Architecture and Specification.* Revision 2.0, July 1995.

[OHJ96] Robert Orfali, Dan Harkey, and Edwards Jeri. *The Essential Distributed Objects Survival Guide*. John Wiley & sns, Inc., 1996.

[OMG96] Object Management Group, Inc. *Mobile Agent Facility Specification*, August 1996. IBM submission.

[OPR96] Randy Otte, Paul Patrick, and Mark Roy. *Understanding Corba - The Common Request Broker Architecture*. Prentice Hall PTR, 1996.

[Pfl89] Charles P. Pfleeger. *Security in Computing*. Prentice Hall, 1989.

[Pos] PostModern Computing. *Orbeline User's guide*.

[RAC94] RACE/TINA-C. *Joint RACE/TINA-C Workshop on Service Engineering*, Brussels, November 1994.

[RC95] Cathy Ring and Philip Carnelly. Distributed objects - creating the virtual mainframe. Technical report, Ovum Limited, 1995.

[RE95] Rosemary Rock-Evans. Middleware - the key to distributed computing. Technical report, Ovum Limited, 1995. edited by Eric Woods.

[RP91] James Rumbaugh and Michael Praha. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[Sei92] Harald Seim. How to handle all the services. *Telektronikk*, (2.92), 1992.

[Sko94] Aasmund Skomedal. *Formal Analysis of mechanisms for Access Control in Distributed Computing Systems*. PhD thesis, NTH, University of Trondheim, 1994.

[Soe93] Lennart Soederberg. Evolving an intelligent architecture for personal telecommunication. *Ericsson Review*, (No. 4,1993 and No. 1,1994), 1993. Ericsson Telecom AB.

[Spi96] Paal Spilling. The telecom research program telerep at unik. *UNIK Aarsrapport 1996*, 1996. http:www.unik.no/ paal/telerep.html.

[SUN90] SUN Microsystems. *Network Programming Guide*, March 1990. Rev A.

[Sun93] Jonas Sundborg. Universal personal teelcommunication (upt) - concept and standardisation. *Ericsson Review*, (No. 4,1993 and No. 1,1994), 1993. Ericsson Telecom AB.

[Sun94] SunSoft. *Universal Networked Objects - ORB 2.0 RFP Submission*. Object Management Group, Inc., Sept 1994. BNR Europe Limited,Expersoft Corporation,IBM,ICL,IONA Technologies and SunSoft, Inc.

[TIN94a] TINA-C. *Engineering Modelling Concepts*, December 1994.

[TIN94b] TINA-C. *Engineering Modelling Concepts (DPE Kernel Specifications)*, November 1994.

[TIN94c] TINA-C. *Requirements Upon TINA-C Architecutre*, December 1994.

[TIN95a] TINA-C. *Computational Modelling Concepts*, January 1995.

[TIN95b] TINA-C. *Connection Management Architecture*, March 1995.

[TIN95c] TINA-C. *Definition of Service Architecture*, February 1995.

[TIN95d] TINA-C. *Distributed Processing Environment (DPE)*, December 1995.

[TIN95e] TINA-C. *Domain types and basic Reference Points in TINA*, May 1995.

[TIN95f] TINA-C. *Information Modelling Concepts*, April 1995.

[TIN95g] TINA-C. *Management Architecture*, March 1995.

[TIN95h] TINA-C. *Object Definition Language*, April 1995.

[TIN95i] TINA-C. *Overall Concepts and Principles of TINA*, February 1995.

[TIN95j] TINA-C. *Service Architecture*, March 1995.

[TIN95k] TINA-C. *Service Component Specifications*, February 1995.

[TIN96a] TINA-C. *DOLMEN*, August 1996. TINA-C Auxiliary project.

[TIN96b] TINA-C. *The Market for Information Services and its demands on TINA-C (TINA-C Enterprise/Business Model)*, March 1996. ver. 2.0.

[TIN96c] TINA-C. *Personal Communications Support in TINA*, June 1996. TINA-C Auxiliary project.

[TIN96d] TINA-C. *Service Architecture 1995*, March 1996.

[TIN96e] TINA-C. *Service Architecture 1996*, December 1996.

[TIN96f] TINA-C. *Service Component Specifications 1995*, March 1996.

[TIN96g] TINA-C. *Unification of Connection/session Graphs, Stream Interfaces and Channel Model*, April 1996. TR_PL.001_1.3_95.

[Tra97] van Dao Tran. Handtering av tjenestedata i en tina dpe. Master's thesis, University of Oslo, Institutt for Informatikk, 1997.

[Wal93] Soeren Wallinder. Implementation of upt - universal personal telecommunication. *Ericsson Review*, (No. 4 1993 and No. 1,1994), 1993. Ericsson Telecom AB.

[Wol94] Rolf Woll. Authentication of upt over muliple networks. Master's thesis, Norwegian Institute of Technology, University of Trondheim, 1994.

ETOP97004

Patent claims

1. Arrangement for improving availability of services in a communications system, especially a telecommunications system, said system comprising distributed hardware and software components which interact in order to provide services to one or more users,
characterized by introducing in said system a mobility transparency, for thereby enabling facilitated application design including inter alia terminal or personal mobility.

2. Arrangement as claimed in claim 1,
characterized in that said mobility transparency is related to all forms of mobility, for example terminal mobility, personal mobility, session mobility, etc., possibly in combination with for example continuous mobility and/or discrete mobility, etc.

3. Arrangement as claimed in claim 1 or 2,
characterized in that said mobility transparency is supported by a functional separation architecture wherein any mobility function(s) is(are) separated both from the associated network layer and the associated application (service) layer.

4. Arrangement as claimed in claim 3,
characterized in that said mobility functions are preferably grouped into a specific layer, i.e. a so-called mobility layer.

5. Arrangement as claimed in any of the preceding claims,
characterized in that said mobility transparency and the support thereof is introduced in any

open distributed proceeding (ODP) system and/or any common request broker architecture (CORBA) system, or similar.

6. Arrangement as claimed in any of the preceding claims,
c h a r a c t e r i z e d i n that in order to realize said mobility transparency and the associated supporting mobility functions, there is introduced a generic mobility system (GMS) which is designed so as to be introduced in said functional separation architecture in a transparent manner.

7. Arrangement as claimed in claim 6,
c h a r a c t e r i z e d i n that said generic mobility system (GMS) is prepared as a middleware which can be customised, configured and installed in any type of network and distribution system to support mobility transparency.

8. Arrangement as claimed in claim 6 or 7,
c h a r a c t e r i z e d i n that said GSM system is configured to cater not only for mobility-unaware applications, by also mobility-aware applications required for for example mobility management, as well as mobility-based applications related to location information held by said GSM.

PATENTSTYRET

Abstract

The Telecommunication Information Networking Architecture is a telecommunications architecture proposed by the TINA Consortium intended to reduce the complexity of the design and implementation of telecommunications applications. The TINA architecture is based on the Open Distributed Processing concepts which aim to consider the whole telecommunications system as a huge mainframe. This is realized through the introduction of a set of distribution transparencies which hides all the complex mechanisms necessary to support distribution.

In this thesis, we show that the defined distribution transparencies are not sufficient to support mobility. We show that it is possible to consider mobility as an additional transparency. In order to support mobility transparency we propose a Functional Separation Architecture where the mobility functions are separated both from the Network layer and the Application (Service) layer and grouped into a layer of its own, called the Mobility layer.

A Generic Mobility System (GMS) is thereafter proposed to realise the Mobility layer. The GMS supports all types of mobility which include terminal mobility, personal mobility and session mobility both in a continuous way and in a discrete way. The GMS is then designed and introduced in the architecture transparently, i.e common applications although supported by the GMS are not aware of its existence. This type of applications is designated as mobility-unaware applications. There are, however, two other types of applications that know about mobility and use it actively. The first type is called mobility-aware application and consists of applications which are required for mobility management such as the user registration. The second type is called mobility-based application and uses actively the location information held by the GMS. Examples are taxi dispatch, fleet management, etc. For these two types of applications the GMS offers an operational interface allowing the query of mobility-related information. The proposed GMS may be implemented and offered as a middleware which can be customised, configured and installed in any type of networks and distributed systems to support the proposed mobility transparency.

An example of the implementation and simulation of the GMS initiated at UNIK, the Center for Technology at Kjeller, is also described in the thesis.

By March 1997 the following three papers have been published as a result of this thesis:

- ***Experiences using corba to build telecommunications applications*** by van Thanh Do and Jan A. Audestad. In *Proceedings of the IFIP/ICCC International Conference on Information Network and Data Communication*, number ISBN 0 412 75750 8. Chapman & Hall, June 1996. Trondheim, Norway.
- ***Making mobility transparent to the applications*** by van Thanh Do and Jan A. Audestad. In *Proceedings of the IEEE VTS 46th Vehicular Technology Conference*, number ISBN 0 7803 3157 5, April 1996. Atlanta, GA USA.
- ***Mobility and Tina*** by van Thanh Do and Jan A. Audestad. In *Proceedings of the TINA 96 Conference*. TINA-C, September 1996. Heidelberg, Germany.

PATENTSTYRET